

XVT-Power++TM

A C++ APPLICATION FRAMEWORK FROM THE #1 COMPANY IN MULTI-PLATFORM DEVELOPMENT TOOLS.

XVT

XVT-Power++

Volume

2

Reference

XVT-POWER++ REFERENCE

XVT - THE PORTABLE GUI DEVELOPMENT SOLUTION

Version 1.1



Copyrights

© 1993 XVT Software Inc. All rights reserved.

The XVT-Power++ application program interface, XVT manuals and technical literature may not be reproduced in any form or by any means except by permission in writing from XVT Software Inc.

XVT and XVT-Power++ are trademarks of XVT Software Inc. Other product names mentioned in this document are trademarks or registered trademarks of their respective holders.

Published By

XVT Software Inc.
Box 18750
Boulder, CO 80308
(303) 443-4223
(303) 443-0969 (FAX)

Credits

Architecture and Design:

Michael Sher

Engineers:

Michael Sher, Ted Kremer, Carol Meier, InfoStructure Inc.

Documentation:

Noreen Garrison, Paul Tepley, Cindi Fisher,
Suanna Jo Schamper

Product Team:

Dave Locke, Michael Sher, Jonathan Auerbach,
Catherine Connor, Peggy Reed, Connie Leserman

Printing History

First Printing November, 1993 XVT-Power++ Version 1.1

This manual is printed on recycled paper by
Continental Graphics, Broomfield, Colorado.



XVT-POWER++

CONTENTS

Introduction	3
XVT-Power++ Classes	7
CApplication	7
CArc	20
CBoss	29
CButtonIcon	35
CCircle	43
CDesktop	49
CDialog	53
CDocument.....	64
CEnvironment	75
CError.....	85
CFixedGrid.....	87
CGlobal	96
CGlobalClassLib	99
CGlobalUser.....	104
CGlue	106
CGrid.....	109
CHWireFrame	126
CIcon	131
CIterator	139
CLine.....	143
CLink.....	150
CList.....	153
CListBox	158
CMem.....	171
CModalWindow	174
CNativeList	181
CNativeSelectList.....	188
CNativeTextEdit.....	195
CNativeView.....	210
COrderedIterator	221

COrderedList	224
COval	229
CPoint	235
CPolygon	240
CPrintMgr	248
CRadioGroup	252
CRect	259
CRectangle	269
CRegularPoly	276
CResourceMgr	284
CScroller	286
CSelectIcon	297
CShape	304
CShellApp	310
CShellDoc	313
CShellWin	316
CSketchPad	322
CSparseArray	331
CSquare	338
CStartup	344
CString	345
CSubview	357
CSwitch	373
CTaskDoc	376
CTaskWin	379
CText	386
CUnits	393
CVWireFrame	404
CVariableGrid	409
CView	420
CVirtualFrame	441
CWindow	450
CWireFrame	465
NButton	476
NCheckBox	481
NLineText	488
NListBox	495
NListButton	501
NListEdit	507
NRadioButton	514
NScrollBar	521
NScrollText	531
NTextEdit	542
NWinScrollBar	552
Index	559

1

INTRODUCTION

This alphabetical presentation of the classes in the XVT-Power++ class hierarchy is designed as a convenient reference for specific information on XVT-Power++'s structure and functionality. It offers a description of each class and its methods and data members, including the protected and private ones. It is targeted towards two groups of readers:

- programmers who want to use XVT-Power++ as an interface for writing portable graphical applications. This group is more interested in how XVT-Power++ works and can be used than in implementation details.
- developers who want to override some XVT-Power++ classes and write new ones. This group needs information about internals (protected and private information) and may need specifics on how the XVT Portability Toolkit and XVT-Power++ are connected.

We assume that you have some basic knowledge of GUI features and programming, a working knowledge of C++, and access to the XVT Portability Toolkit documentation.

We have sought to minimize the need to refer to the XVT Portability Toolkit manual and thus show the XVT Portability Toolkit definitions where appropriate. We include tables of the XVT Portability Toolkit options that are pertinent to various classes—for environment settings, window types, unsigned attributes, and so on. Also with convenience in mind, we have chosen to minimize cross references and to make each discussion of a class as complete and as independent as possible. The result is a lot of repetition, but you can find almost everything you need to know about a given class in one place. The discussion of each class includes:

- a description of the class and its usage

- a descriptive listing of its data members
- descriptions of methods specific to the class
- descriptions of methods that are inherited and overridden
- a listing of methods that are inherited but not overridden
- the header (.h) file.

Note that the index to this manual contains a comprehensive listing of the tasks that can be performed for each class.

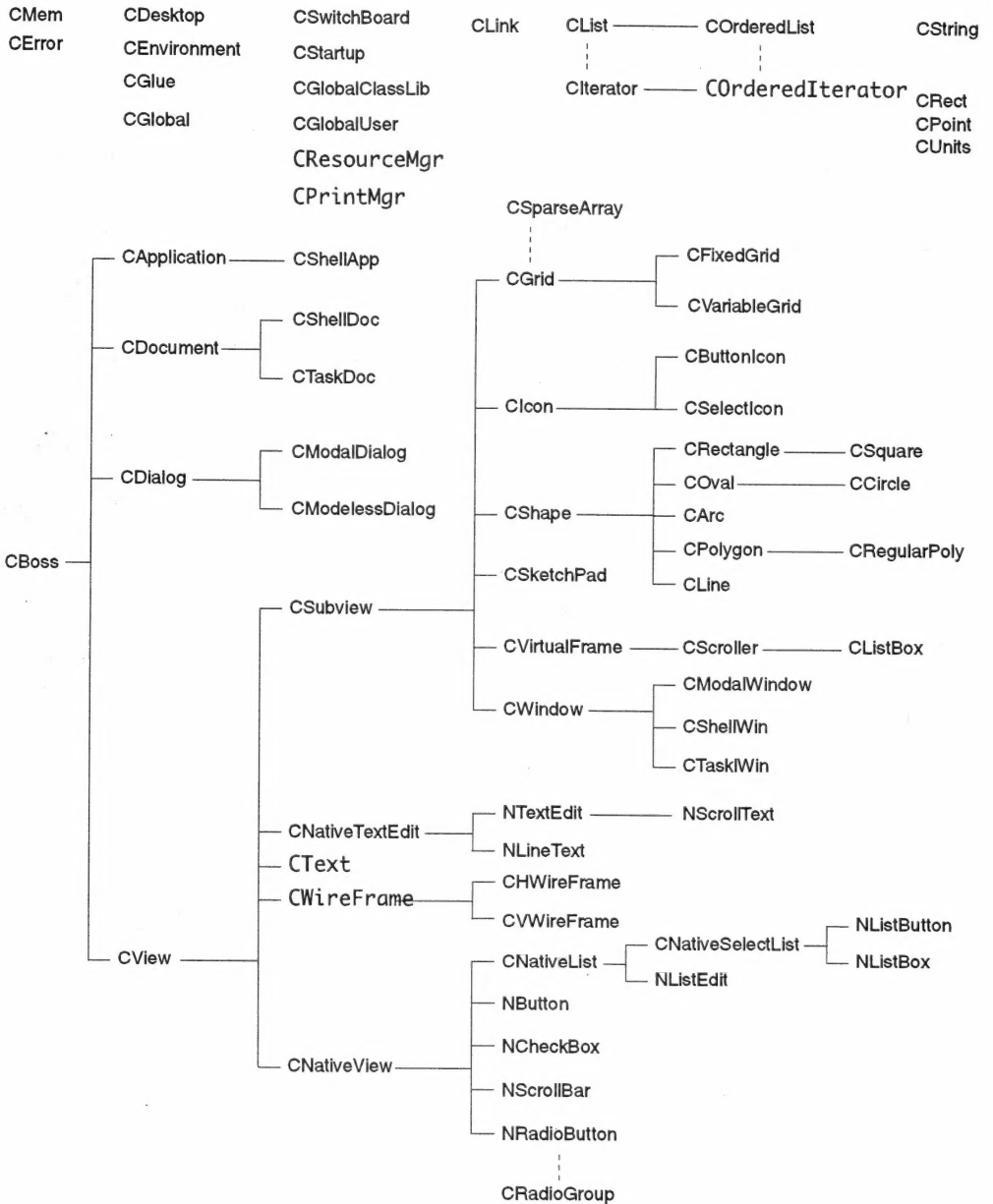


Figure 1. XVT-Power++ Hierarchy

XVT-Power++ Classes	Purpose
Event Handling: CSwitchBoard	Receive and dispatch events to all appropriate objects.
Application Framework: CApplication, CDocument, CView	Divide the work in an application into program control, data storage and management, and data display.
Shell: CShellApp, CShellDoc, CShellWin	Override the basic application framework, providing an easy way to get started.
Controls: CNativeView, NButton, NCheckBox, NScrollBar, NRadioButton, CRadioGroup, MWinScrollBar, CNativeList, CNativeSelectList, NListEdit, NListButton, NListBox	Correspond to the XVT Portability Toolkit control objects, providing the means of communication between the application and the user operating the mouse
Subviews: CSubview	Provide subviews that can nest recursively within other subviews and propagate events to all of their subviews.
Moving and Sizing: CWireFrame, CSketchPad, CGlue	Draw and glue views, size and drag views, establish local coordinate systems and clipping properties.
Icons: CIcon, CButtonIcon, CSelectIcon	Draw icon resources and inherit all view functionality.
Panning: CVirtualFrame, CScroller, CListBox, CScrollText	Serve as containers in which other objects are displayed and manipulated.
Text: CNativeTextEdit, NTextEdit, NLineText, NScrollText, CText	Provide extended text editing capabilities, including field validation, sizing, dragging, cut, copy, paste, automatic selection, text setting, and text retrieval.
Shapes: CShape, C Oval, C Circle, C Rectangle, C Square, C Arc, C Polygon, C RegularPoly, C Line	Draw shapes from a set of shapes that have the full functionality of all views and that can nest within other views, receiving and passing events.
Windows and Dialogs: CWindow, CModalWindow, CTaskWin and CTaskDoc, CDialogWindow, CModalDialog, CModelessDialog	Handle all window events: mouse clicks and drags, updating, menubars, sizing, and so forth.
Window Management and Layout: CDesktop, CGlue, CGrid, CFixedGrid, CVariableGrid, CSparseArray	Place windows and/or views and determine stacking order, sizing, and gluing; arrange views into rows and columns
Data Structures: CList, CLink, CIterator, COrderedList, COrderedIterator, CString, CRect, CPoint, CUnits	Provide many data structuring features used by GUI applications, such as rectangle intersection, union, and translation.
Utilities: CBoss, CGlobalClassLib, CGlobalUser, CStartup, CEnvironment, CResourceMgr, CMem, CError, CGlobal	Perform utility functions such as keeping track of class library and application global data.

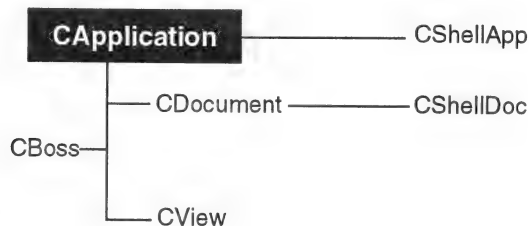
Figure 2. XVT-Power++ Classes Grouped by Task

2

XVT-POWER++ CLASSES

This chapter describes the XVT-Power++ classes.

CApplication



Description

Each application has one CApplication object, which is the highest object in the application framework hierarchy. This object is the first to be instantiated, the first to receive the thread of control, and the last to be destroyed. It is also the last object to receive events when no other objects can handle them. The CApplication object maintains a list of CDocument objects and creates the global desktop and global environment. CApplication is an abstract class and cannot be instantiated; you must derive an instance from it instead.

Heritage

Superclass: CBoss

Subclass: CSHELLApp

Usage

You must create a derived instance of this abstract class. There can be only one instance of CApplication within a running application. The application is normally created inside of main, as shown in the following example:

```
void main(int argc, char* argv[])
{
    CHardHatApp testApp;
    testApp.Go(argc, argv, MENU_BAR_RID, 0, "BaseName",
               "Application Name", "Task Window Title");
}
```

Refer to the Go method for more information on each of the parameters passed in. Once the application is created and the Go method is called in the application, the Go function never returns to the user. Thus, you should set up everything before calling Go.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

CList	*itsDocuments	the list of documents maintained by the application
CEnvironment	*itsEnvironment;	the global default environment
static BOOLEAN	itIsInstantiated	assures that only one CApplication object exists and is initially set to FALSE
CSwitchBoard	*itsSwitchBoard;	assures that only one CApplication object exists
PRINT_RCD	*itsPrintRecord;	the application's print record

Friends

friend class	CDocument	a document that can call the protected methods and add or remove itself from the application object's CList.
--------------	-----------	--

Public Methods

Constructor, Initializer, and Destructor Methods

CApplication(void);

A constructor. It sets up the application object and acts as a safeguard, ensuring that only one application object is instantiated per application.

The application should be created inside of main. This constructor just sets up some global objects and some initial data structures—as should any derived class constructors. The constructor should take care of only these tasks. No other object from the XVT-Power++ application framework should be created until the application receives a StartUp event.

CApplication(const CApplication& theApplication);

A constructor. It guarantees that no application object is instantiated twice because there can be only one instance of a CApplication object per application.

virtual ~CApplication(void);

The destructor, which is called through application cleanup. When it is called, the application deletes all of its documents and deletes itself.

Override:

The user is responsible for deleting the CGlobalUser object. While the destructor for the application cleans up any global objects created by XVT-Power++, it is the responsibility of the derived constructor to delete any objects created by the derived application. If a global user object is created, be sure to delete it and set the GU pointer to NULL as shown here:

```
CMyApplication::~CMyApplication{void}
{
    delete GU;

    //Delete leaves GU undefined; set to NULL
    before terminating GU=NULL;

    : : :
}
```

XVT Portability Toolkit Start-up Methods

```
virtual void Go(int argc, char *argv[], short  
theMenuBarId, short theAboutBoxId, char  
*theBaseName, char *theApplicationName, char  
*theTaskTitle);
```

A method that takes control of the application from main. It is called inside main on the application object that is created there. The parameters argc and argv are the standard C parameters that are passed to the main function. theMenuBarId is the resource ID number of the task menubar in XVT Portability Toolkit's URL. Similarly, theAboutBoxId is the resource ID number of the application's default About box. theBaseName is the application's file name. That is, it is the base file name, without extension, that the XVT Portability Toolkit uses when looking for .hlp files, .frl files, and .uid files.

theApplicationName specifies the name that is prepended to the titles of document windows when the title of the document is set. Finally, theTaskTitle is the title of the task window for platforms where the task window has a physical representation.

After Go is called, control is not returned to main, so no code should follow Go. In fact, there should be very little code before Go is called. Very little should be done in main other than creating the application and invoking Go.

```
virtual void StartUp(void);
```

A virtual method that handles start-up activities for the application. You can override it in your own particular application class. The first thing you must do is call the inherited StartUp method. CApplication::StartUp is called before anything else inside of StartUp. After that, you specify what you want your application to do upon start-up: create windows, open documents, connect to a database, and so on.

```
virtual void ShutDown(void);
```

A virtual method that handles the shut-down activities of an application. You can override in your application class. The first thing it should do is call the inherited ShutDown method. After that, you specify the actions you want it to take upon application shut-down. Normally, you do not have to perform any actions related to XVT-Power++ objects. For example, if windows are up, they are closed and deleted automatically through XVT-Power++. The objects inside these windows are also deleted automatically. What you must do is close any files, disconnect from a database, and perform any other application-dependant tasks.

Command Events

```
virtual void DoCommand(long theCommand, void*
    theData=NULL);
```

A method that handles command events. It can be called either directly or by a document that has propagated the event upwards to the application. The void pointer theData can be used to pass any user-defined structure to the DoCommand method.

Menu Events

```
virtual void DoMenuCommand(MENU_TAG
    theMenuItem, BOOLEAN isShiftKey, BOOLEAN
    isControlKey);
```

Handles the events that occur when the user selects an item from the menubar. DoMenuCommand first goes the window from which the selection was made. If the window cannot handle the command, it sends it to its document. The event chains upwards to the application. theMenuItem is the menu item number of the command, starting at one. Disabled commands and dividing lines count in the numbering. isShiftKey and isControlKey specify whether the mouse is used with the SHIFT key or CONTROL key.

DoMenuCommand has the following predefined actions it takes upon a predefined menu, the File menu. The methods called by these predefined functions are described in the next section, called "File Menu Item Events."

M_FILE_QUIT

Calls the application's DoClose method. DoClose in turn queries every document for quitting and returns TRUE if all documents agree to close. If DoClose returns TRUE, the application terminates.

M_FILE_OPEN

Calls the default DoOpen method when the user selects "Open" from the menu. See DoOpen.

M_FILE_CLOSE

Calls the default DoClose method when the user selects "Close" from the menu. See DoClose.

M_FILE_SAVE

Calls the default DoSave method when the user selects "Save" from the menu. See DoSave.

M_FILE_SAVE_AS

Calls the default DoSaveAs method when the user selects "Save As" from the menu. See DoSaveAs.

M_FILE_ABOUT

Calls the default DoAboutBox method when the user selects “About Box” from the menu. See DoAboutBox.

M_FILE_PG_SETUP

Calls the default DoPageSetup method when the user selects Page Setup from the menu. See DoPageSetup.

M_FILE_PRINT

Calls the default DoPrint method when the user selects “Print” from the menu. See DoPrint.

M_FILE_NEW

Calls the default method DoNew when the user selects “New” from the menu. See DoNew.

virtual void ChangeFont(FONT theFont, FONT_PART thePart);

When the user selects a font from the menubar, ChangeFont sends a font object, theFont, that indicates the current state of the Font/Style menu. The specific font property that was changed is indicated in thePart, which is of type FONT_PART, an XVT Portability Toolkit variable that is defined as follows:

```
typedef enum{          /* symbol for part of font desc.*/
    F_STYLE;           /* style (bold, italic, ect.)*/
    F_FAMILY;          /* FAMILY (Times, etc.) */
    F_SIZE;            /* point size part*/
} FONT_PART;
```

ChangeFont is an event that chains upwards from a window. This event goes directly to the application if no windows are up. This method sets the font for the global application environment. You should not call it directly; instead, call SetEnvironment to change the application’s font.

virtual void SetUpMenus(void);

Automatically called to initialize menus, this method allows you to set up menus, activating the File menu, creating a new menu, and so forth.

By default this method is empty. You must override it to provide information on setting up the initial menu default.

virtual void UpdateMenus(void);

Enables the application, based on some state, to set up the appropriate menus—checking menu items, unchecking them, and so forth. This method is called when a window is selected. When a window object updates its menus, the update event can

pass up to its parent and chain upward, stopping at any point, till it reaches the CApplication object.

The default mechanism does nothing. You must override UpdateMenus to change menu states so that they reflect the state of the program.

File Menu Item Events

Each of the methods described in this section is called when the user selects an item from the File menu.

virtual void DoAboutBox(void);

Displays an About box, either automatically through an explicit call or in response to a user selection from a menu. You may want to override this method to get your own animated About box using an XVT-Power++ type of window instead of an XVT Portability Toolkit type of window. In this case, do not call an inherited object, which brings up the standard About box.

virtual BOOLEAN DoClose(void);

Sends a DoClose message to all of the documents of the application. The documents can query the user before closing and return TRUE if the *close* operation succeeds. If all documents close, this method returns TRUE; otherwise, it returns FALSE to indicate that some documents are still open. This method is called internally when the user selects "Quit" from the File menu or closes the task window.

virtual BOOLEAN DoOpen(void);

When the user selects "Open" from the File menu, the application object creates a new document and then calls this method, which opens the document. You must provide the appropriate action(s) in your derived class by overriding DoOpen.

Override:

Instantiate a new document and tell that document to open. This is where you put code for invoking an "Open" dialog box that gives access to some data. See CShellApp for an example of DoOpen.

virtual BOOLEAN DoNew(void);

Creates a new document when there is no data, unlike Open which gives access to existing data. You must provide the appropriate action(s) in your derived class by overriding this method.

Override:

Instantiate a new document and tell that document to open. This is where you put code for invoking an “Open” dialog box that gives access to some data. See CShellApp for an example of DoNew.

virtual BOOLEAN DoPageSetUp(void);

A method that is called when the user selects the “Page SetUp” menu item. DoPageSetUp calls the XVT Portability Toolkit `page_setup_dlg` function, which sets up the standard page setup dialog. This method returns a Boolean value of TRUE or FALSE, depending on whether the page set-up dialog box succeeded. The page set-up information is contained in the global class library; it has the global `G ->`. Its print record is the information that is set by the page set-up dialog box.

The page set-up dialog box is not invoked on all platforms—currently it is operative on the Macintosh and in Windows but not in Motif, which does not have a “Page SetUp” option on the File menu.

virtual BOOLEAN DoPrint(void);

Takes all the open windows in the application, dumps each window onto a separate page, and sends it to the printer.

virtual BOOLEAN DoSave(void);

Calls the CDocument DoSave method on any documents that need saving. This method returns a Boolean value of TRUE if all *save* operations are successful.

virtual BOOLEAN DoSaveAs(void);

Calls the CDocument DoSaveAs method on any documents that need saving. DoSaveAs returns a Boolean value of TRUE if all *save* operations are successful;

Keyboard Events**virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);**

A method that is called when there is a keyboard event. When the user presses a key on the keyboard, the XVT Portability Toolkit sends an `E_CHAR` event to the switchboard, which passes this event to the selected window. The window checks to see if one of its subviews is selected, and, if so, passes the event to the appropriate subview. If no subview is selected, the keyboard event propagates upwards from the window to the document

and finally to the application. A keyboard event goes directly to the application if no windows are open.

`theKey` is the ASCII number of the character key that was pressed; `isShiftKey` and `isControlKey` indicate whether the SHIFT key or CONTROL key was pressed in conjunction with the character key. You can override `DoKey` to do something specific to a keyboard action within the application. The default mechanism does nothing.

Document Management Methods

```
virtual CDocument* FindDocument(int theId)  
const;
```

Searches for a document using the ID number assigned to it upon creation and stored in `CList`, the document ID list maintained by the application.

```
virtual void CloseAll(void);
```

Closes and deletes all the documents in the application. Unlike `DoClose`, this method disregards whether documents need saving or not.

```
int GetNumDocuments(void);
```

Returns the number of documents in the application.

```
virtual CList* GetDocuments(void) const;
```

Returns a list of the application's documents.

Getting Subobjects

```
virtual const CList* GetSubObjects(void) const;
```

A pure virtual method that must be defined by any class derived from `CBoss`, specifically, `CApplication`, `CDocument`, `CView`, and `CSubview`. It returns a list of any subobjects associated with a given object. For example, the application would return a list of all its documents, a document would return a list of all its windows, a window would return a list of all its enclosed views, a subview would return a list of all its enclosed views, and so on.

Environment Methods

```
virtual const CEnvironment*  
GetEnvironment(void) const;
```

Returns the application's global environment.

```
virtual void DoSetEnvironment(const  
    CEnvironment& theNewEnvironment, BOOLEAN  
    isUpdate = FALSE);
```

Sets the application's global environment to the given environment, `theNewEnvironment`. In addition it sends an "update environment" message to all documents that depend on the global environment.

```
virtual void SetEnvironment(const CEnvironment&  
    theNewEnvironment);
```

Sets the environment for a view, using `theNewEnvironment` that is passed to it. By default, views share their enclosure's environment. However, as soon as you use `SetEnvironment` to give a view an environment of its own, the view uses that environment instead of the shared environment.

When the global environment is changed, all views sharing that environment get a `SetEnvironment` message with the `isUpdate` parameter set to `TRUE`. The `isUpdate` parameter indicates whether this `SetEnvironment` message is simply an update message or whether it is a "real" `SetEnvironment` message meaning that this particular view should set its own environment to the new environment.

Printing Methods

```
virtual PRINT_RCD* GetPrintRecord(void) const;
```

Returns the application's global print record, which supplies the necessary information about the printing set-up, such as the page set-up, the current printer, and so on. On some platforms, it may also keep track of information on the job status as it progresses. Each application has a global print record that is available through the `CApplication` class by means of this method.

Event Handler Methods

```
virtual void DoTimer(long theTimerId);
```

A method that is called when the XVT Portability Toolkit generates a timer (`E_TIMER`) event. You set a timer using the XVT Portability Toolkit's `set_timer` function, which takes a window and a time interval (in milliseconds) and returns an ID number for the timer. This is the ID number that is passed to `theTimerId` of `DoTimer`. When the timer generates a timer event, `CSwitchBoard` passes it to the XVT Portability Toolkit-designated window via `DoTimer`. The window passes it on to its

selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

```
virtual void DoUser(long theUserId, void* theData);
```

A method that is called when the XVT Portability Toolkit generates a user (E_USER) event. As noted in the *XVT Programmer's Guide*, user events allow you to pass custom events to windows and dialogs. theUserId is the ID number that designates a particular kind of user event. The void pointer theData can be used to pass any user-defined structure to DoUser, just as you would for a DoCommand. As with timer events, CSwitchBoard passes a user event to a designated window. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

Protected Methods

```
BOOLEAN IApplication(CGlobalUser *theGlobalUser);
```

When you develop an XVT-Power++ application, certain objects, or variables, need to be globally accessible to any object in the hierarchy. The CGlobalUser object provides a central place to store global objects for user-derived application objects in a concise, uniform way. IApplication allows you to set up the global user for the user-derived application object. IApplication returns a Boolean value of TRUE if it has succeeded. See CGlobalUser.

```
virtual void AddDocument(CDocument *theDocument);
```

Adds a document to the application.

```
virtual void RemoveDocument(CDocument *theDocument);
```

Removes a document from the application.

Private Methods

None.

Overrides:

Nothing must be overridden, but you may want to write

new event handling methods since most of the ones provided are empty.

Methods Inherited But Not Overridden

From CBoss

```
virtual CUnits* GetUnits(void) const;
virtual void SetUnits(CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

Summary: CApplication.h

```
#ifndef CApplication_H
#define CApplication_H

#include "CBoss.h"

class CDesktop;
class CDocument;
class CList;
class CEnvironment;
class CSwitchBoard;

class CApplication :public CBoss
{
public:
    // construct & destruct & init
    CApplication(void);
    CApplication(const CApplication& theApplication);
    virtual ~CApplication(void);

    // XVT Portability Toolkit startup
    virtual void Go(int argc, char *argv[],short theMenuBarId,
        short theAboutBoxId, char *theBaseName,
        char *theApplicationName,char *theTaskTitle);
    virtual void StartUp(void);
    virtual void ShutDown(void);

    // Command events
    virtual void DoCommand(long theCommand, void* theData=NULL);

    // File Menu item events
    virtual void DoAboutBox(void);
    virtual BOOLEAN DoClose(void);
    virtual BOOLEAN DoOpen(void);
    virtual BOOLEAN DoNew(void);
    virtual BOOLEAN DoPageSetUp(void);
    virtual BOOLEAN DoPrint(void);
    virtual BOOLEAN DoSave(void);
    virtual BOOLEAN DoSaveAs(void);

    // Keyboard events:
    virtual void DoKey(int theKey, BOOLEAN isShiftKey,
        BOOLEAN isControlKey);
```

```

// Menu events:
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey,
                           BOOLEAN isControlKey);
virtual void ChangeFont(FONT theFont, FONT_PART thePart);
virtual void SetUpMenus(void);
virtual void UpdateMenus(void);

// Document management:
virtual CDocument* FindDocument(int theId) const;
virtual void CloseAll(void);
int GetNumDocuments(void);
virtual CList* GetDocuments(void) const;

// Other:
virtual const CList* GetSubObjects(void) const;
virtual const CEnvironment* GetEnvironment(void) const;
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment,
                              BOOLEAN isUpdate = FALSE);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment);

virtual PRINT_RCD* GetPrintRecord(void) const;

// Other event handler methods:
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);

protected:

BOOLEAN IApplication(CGlobalUser* theGlobalUser);
virtual void AddDocument(CDocument* theDocument);
virtual void RemoveDocument(CDocument* theDocument);

private:

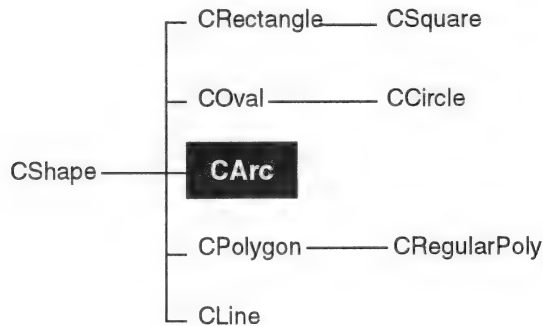
friend class CDocument; // Allow documents to self-add and self-remove

CList* itsDocuments; // the list of the documents
CEnvironment *itsEnvironment; //the global default environment
static BOOLEAN itIsInstantiated; // assure only one CApp exists
CSwitchBoard *itsSwitchBoard; //the global default event handler
PRINT_RCD *itsPrintRecord; // the application's print record
};

#endif CApplication_H

```

CArc



Description

CArc objects paint an arc inside a view. This arc, which is a section of the perimeter of an oval, is drawn counterclockwise along the oval from a given angle to another.

Heritage

Superclass: CShape

Usage

Create a CArc object and initialize it. Like all CShape classes, this class inherits all the properties of CSubview, including stickiness and the ability to nest other views.

Environment

The line of the arc is drawn with the *pen*, and its interior is painted with the *brush*. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width.

Public Data Members

None.

Private Data Members

None.

Private Data Members

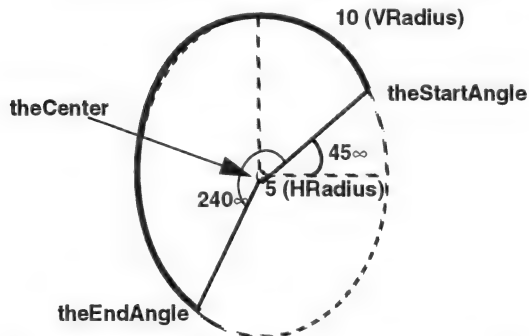
BOOLEAN	itIsFilled;	if TRUE, an interior is painted
UNITS	itsStartX,itsStartY;	the starting point coordinates
UNITS	itsEndX,itsEndY;	the ending point coordinates
double	itsStartAngle,itsEndAngle;	

Public Methods

Constructor, Destructor, and Initializer Methods

```
CArc(CSubview* theEnclosure, const CPoint&
theCenter, UNITS theHRadius, UNITS
theVRadius, double theStartAngle=0, double
theEndAngle=360);
```

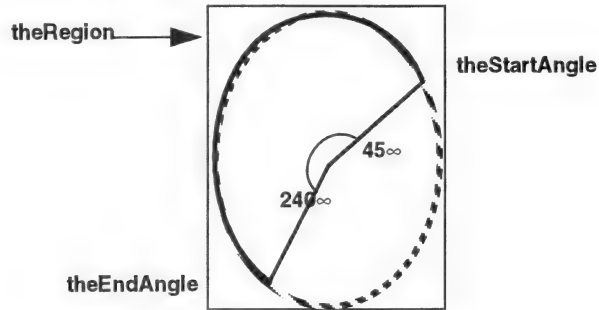
A constructor. theEnclosure is a pointer to the subview that will contain the arc. theCenter is the center point for the arc (a CPoint that is relative to the arc's enclosure). This method also requires a horizontal radius and a vertical radius. The arc is drawn from theStartAngle to theEndAngle, which are measured in degrees with a valid range of 0° to 360°.



```
CArc(CSubview* theEnclosure, const CRect&
theRegion, double theStartAngle=0, double
theEndAngle=360);
```

A constructor. theEnclosure is a pointer to the subview that will contain the CArc object. theRegion is a coordinate location, local to the theEnclosure, that is used to place the CArc object. The arc is drawn from theStartAngle to

theEndAngle, which are measured in degrees with a valid range of 0° to 360° .



```
CArc(const CArc& theArc);
```

A copy constructor that creates a new CArc object with the same enclosure, color, visibility attributes, enabled/disabled attributes, environment, and so on as the original CArc object. However, any views nested within the original CArc object are not copied.

```
CArc& operator=(const CArc& theArc);
```

An assignment operator. It copies the attributes of the original CArc object, creating a new CArc object that has the same color, glue, environment, visibility state, and so on. However, any shapes nested within the original CArc object are not copied.

```
virtual ~CArc(void);
```

The destructor. It cleans up after the arc and deletes any views nested within it.

```
BOOLEAN IArc(double theStartAngle = 0,  
              double theEndAngle = 360,  
              BOOLEAN isFilled   = FALSE,  
              BOOLEAN isVisible  = TRUE,  
              GLUETYPE theGlue   = NULLSTICKY);
```

The initializer. It takes a starting angle and an ending angle on an imaginary oval. By default, the CArc object has no interior fill, is visible, and uses the glue type NULLSTICKY.

```
virtual void Draw(const CRect&  
                  theClippingRegion);
```

Takes care of any drawing the arc must do. theClippingRegion is the part of the arc that needs to be drawn and is in global, window-relative coordinates.

Utility Methods

virtual void SetFilled(BOOLEAN isFilled);

Takes a Boolean value. If it is set to TRUE, the arc has an interior fill (draws a pie); if it is set to FALSE, the arc does not have a fill.

virtual BOOLEAN IsFilled(void);

Returns a Boolean value indicating whether the arc has a fill.

virtual void SetAngles(double theStartingAngle, double theEndingAngle);

Sets the starting and ending angles of the arc on an imaginary oval. The arc is drawn from theStartAngle to theEndAngle, which are measured in degrees with a valid range of 0° to 360°.

virtual double GetStartingAngle(void);

Returns a number, which is the starting angle of the arc on an imaginary circle. theStartAngle is measured in degrees with a valid range of 0° to 360°.

virtual double GetEndingAngle(void);

Returns a number, which is the ending angle of the arc on an imaginary circle. theEndAngle is measured in degrees with a valid range of 0° to 360°.

Inherited Utility Methods

virtual void Size(const CRect& theNewSize);

Sizes the arc according to the coordinates of theNewSize. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, theNewSize, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated.

Private Methods

void SetDrawingPoints(const CPoint& theStartingPoint, const CPoint& theEndingPoint);

Translates the coordinates and angles of an arc into points on the screen.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);  
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);  
virtual CUnits* GetUnits(void) const;  
virtual void SetUnits (CUnits* theCoordinateUnits);  
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);  
virtual void Deactivate(void);  
virtual void Disable(void);  
virtual void DoCommand(long theCommand, void* theData=NULL);  
virtual BOOLEAN DoPrint(const CRect& theRegion) const;  
virtual void DoTimer(long theTimerId);  
virtual void DoUser(long theUserId, void* theData);  
virtual void Draw(void);  
virtual void Enable(void);  
virtual CView* FindHitView(const CPoint& theLocation) const;  
virtual CRect GetClippedFrame(void) const;  
virtual long GetCommand(void) const;  
virtual CWindow* GetCWindow(void) const;  
virtual long GetDoubleCommand(void) const;  
virtual CSubview* GetEnclosure(void);  
virtual const CEnvironment* GetEnvironment(void) const;  
virtual CRect GetFrame(void) const;  
virtual CRect GetGlobalFrame(void) const;  
virtual CPoint GetGlobalOrigin(void) const;  
virtual GLUETYPE GetGlue(void) const;  
virtual int GetId(void) const;  
virtual CRect GetLocalFrame(void) const;  
virtual CPoint GetOrigin(void) const;  
virtual const CString GetTitle(void) const;
```

```
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;

```

```

virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CShape

None.

Summary: CArc.h

```

#ifndef CArc_H
#define CArc_H

#include "CShape.h"

class CArc : public CShape
{
public:
    // construct, destroy, initialize:
    CArc(CSubview* theEnclosure, const CPoint& theCenter,
        UNITS theHRadius, UNITS theVRadius,
        double theStartAngle=0, double theEndAngle=360);

    CArc(CSubview* theEnclosure, const CRect& theRegion,
        double theStartAngle=0, double theEndAngle=360);

    CArc(const CArc& theArc);
    CArc& operator=(const CArc& theArc);
    virtual ~CArc(void);

    BOOLEAN IArc(double theStartAngle = 0,
        double theEndAngle = 360,
        BOOLEAN isFilled = FALSE,
        BOOLEAN isVisible = TRUE,
        GLUETYPE theGlue = NULLSTICKY);

    virtual void Draw(const CRect& theClippingRegion);

    // util
    virtual void SetFilled(BOOLEAN isFilled);
    virtual BOOLEAN IsFilled(void);

    virtual void SetAngles(double theStartingAngle, double theEndingAngle);
    virtual double GetStartingAngle(void);
    virtual double GetEndingAngle(void);

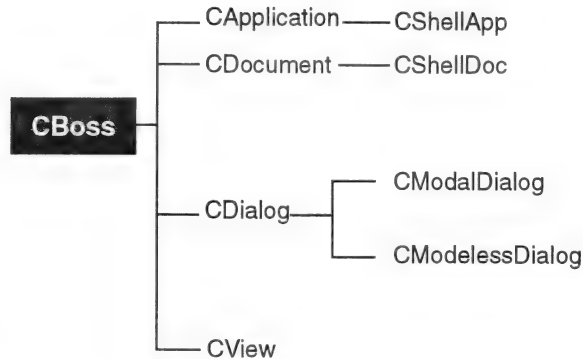
    // Inherited utility:
    virtual void Size(const CRect& theNewSize);

private:
    BOOLEAN itIsFilled; // If TRUE, an interior is painted
    UNITS itsStartX, itsStartY; // The starting point coordinates
    UNITS itsEndX, itsEndY; // The ending point coordinates
    double itsStartAngle, itsEndAngle;

```

```
        void SetDrawingPoints(const CPoint& theStartingPoint,  
                               const CPoint& theEndingPoint);  
};  
#endif CArc_H
```

CBoss



Description

CBoss provides access to global objects and global data, and it ties in classes under a common memory manager. Also, CBoss provides basic internal functionality to the classes, such as the basis for the event and message passing structure. An abstract class, it is used mainly for purposes of inheritance. All classes in the XVT-Power++ application framework (consisting of view, document, and application branches) inherit from CBoss.

Heritage

Superclass: none

Subclasses: CApplication, CDocument, CView

Usage

This class is never instantiated. Typically, CBoss is used through the CApplication class, which initializes and creates its static globals. You should never need to derive an object directly from CBoss, but you could do it to extend the class library.

Public Data Members

As a common ancestor, CBoss acts as a holder of all static global variables that are shared among the application objects. These static globals are for the class library and for users who are creating applications based on XVT-Power++.

CBoss has two public data members, which are pointers. Anything that inherits through the CBoss object has access to the global data through one of these pointers. Both of these static pointers are initialized to NULL, so you must set these pointers to point to an actual object.

<code>static CGlobalClassLib *G;</code>	XVT-Power++ global objects
<code>static CGlobalUser *GU;</code>	user-specific global objects

Protected Data Members

`CUnits *itsUnits;` the logical coordinates of the object

Private Data Members

None.

Friends

<code>friend class CApplication;</code>	allow application to set G
<code>friend class CUnits;</code>	allow units to call <code>UpdateUnits</code>

Public Methods

Event Hooks

CBoss has three methods for event hooks that are located in objects within the application framework hierarchy. These are the three primary events that every application, document, and view object can handle.

```
virtual void DoCommand(long theCommand, void*
    theData=NULL);
```

A DoCommand event starts at some object and then chains upward through the application framework event hierarchy, stopping at any point. These events can be passed all the way up to the application object. The path a DoCommand travels is based on a supervisor relationship. It follows the rule of looking for its boss, which is the object that is in charge of it. The void pointer theData can be used to pass any user-defined structure to the DoCommand method.

```
virtual void DoMenuCommand(MENU_TAG
    theMenuItem, BOOLEAN isShiftKey, BOOLEAN
    isControlKey);
```


Handles the events that occur when the user selects an item from the menubar. DoMenuCommand first goes to the window from which the selection was made. If the window cannot handle the command, it sends the command to its document. The event chains upward, stopping at any point, to the application. theMenuItem is the menu item number of the command, starting at one. Disabled commands and dividing lines count in the numbering. isShiftKey and isControlKey specify whether the mouse is used with the SHIFT key or CONTROL key. Note that this method is not called for menubar font selections; instead the ChangeFont method is called.

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

When the user changes a font, ChangeFont sends a font object, theFont, that indicates the current state of the Font/Style menu. The specific font property that was changed is indicated in theChange, which is of type FONT_PART, an XVT Portability Toolkit variable.

ChangeFont is an event that chains upward from some view, as does DoMenuCommand. You must override this method to define actions for updating fonts. CWindow, CDocument, and CApplication all have the appropriate overridden methods.

Event Handler Methods

```
virtual void DoTimer(long theTimerId);
```

A method that is called when the XVT Portability Toolkit generates a timer (E_TIMER) event. You set a timer using the XVT Portability Toolkit's set_timer function, which takes a window and a time interval (in milliseconds) and returns an ID number for the timer. This is the ID number that is passed to theTimerId of DoTimer. When the timer generates a timer event, CSwitchBoard passes it to the XVT Portability Toolkit-designated window via DoTimer. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

```
virtual void DoUser(long theUserId, void* theData);
```

A method that is called when the XVT Portability Toolkit generates a user (E_USER) event. As noted in the *XVT Programmer's Guide*, user events allow you to pass custom events to windows and dialogs. theUserId is the ID number

that designates a particular kind of user event. The void pointer `theData` can be used to pass any user-defined structure to `DoUser`, just as you would for a `DoCommand`. As with timer events, `CSwitchBoard` passes a user event to a designated window. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

Coordinate Logical Units

```
virtual CUnits* GetUnits(void) const;
```

Returns a pointer to the object's `CUnits` object. That is, this method returns the logical units for an object of type `CBoss`. The logical units are mapped out to the physical device on which you are displaying your information. We use the term "physical device" because the units can be mapped not only to a screen display but also to a printer display. If they are mapped to the screen, the logical units are translated into pixels; if they are mapped to a printer, then the logical units are translated into printer units. See `CUnits`.

```
virtual void SetUnits(CUnits*  
theCoordinateUnits);
```

Sets the units of a `CBoss` object. The units are logical units that are mapped either to the screen or to the printer. Logical units can be measured in inches, centimeters, characters, pixels, or a user-defined measurement. In XVT-Power++, the default unit is pixels. See `CUnits`.

Sub-Object Management

```
virtual const CList* GetSubObjects(void) const  
= NULL;
```

A pure virtual method that must be defined by any class derived from `CBoss`, specifically, `CApplication`, `CDocument`, `CView`, and `CSubview`. It returns a list of any subobjects associated with a given object. For example, the application would return a list of all its documents, a document would return a list of all its windows, a window would return a list of all its enclosed views, and so on.

Protected Methods

Constructor and Initializer Methods

```
CBoss (void);
```

The default constructor.

```
CBoss (const CBoss& theBoss);
```

The copy constructor. It takes a constant and a reference to a boss. It is primarily used for initialization through another object and should be called by the copy constructors of derived classes.

```
CBoss& operator=(const CBoss& theBoss);
```

Defines the equal operator for classes that derive from CBoss. This is a shallow copy; it does not copy the pointers of all the objects inside it. This method should be called by the equal operator of derived classes.

```
virtual ~CBoss(void);
```

The destructor. It cleans up after CBoss.

```
BOOLEAN IBoss(CGlobalUser *theGlobalUser);
```

When a global user class is created, this initializer is used to inform CBoss of the existence of the user-specified globals. theGlobalUser is a pointer to an object of type CGlobalUser that contains global utilities for a user application.

```
virtual void UpdateUnits(CUnits* theUnits);
```

Updates the CUnits object indicated by theUnits, which is the object owned by the CBoss object. The owner updates itself and propagates the update message to any of its child objects. For example, the application would update all of its documents, the documents would update all of their windows, and so on, but only if they are sharing the same CUnits object. Basically, the owner of a CUnits object is the topmost object in the hierarchy that is using the CUnits object.

Private Methods

```
BOOLEAN IBoss(CGlobalClassLib  
*theGlobalClassLib);
```

When a global class library is created, this initializer is used to inform CBoss of the existence of the globals. theGlobalClassLib is a pointer to an object of type CGlobalClassLib that contains global utilities for XVT-Power++.

Summary: CBoss.h

```
#ifndef CBoss_H  
#define CBoss_H
```

```

#include "xvt.h"
#include "PwrNames.h"
#include "PwrDef.h"
#include "CMem.h"

#include CGlobalClassLib_i
#include CGlobal_i

class CGlobalUser;
class CUnits;
class CList;

class CBoss
{
public:
    static CGlobalClassLib *G; // PWR library global objects
    static CGlobalUser *GU; // user specific global objects

    // Event hooks:
    virtual void DoCommand(long theCommand, void* theData=NULL);
    virtual void DoMenuCommand(MENU_TAG theMenuItem,
        BOOLEAN isShiftKey, BOOLEAN isControlKey);
    virtual void ChangeFont(FONT theFont, FONT_PART theChange);

    // Other event handler methods:
    virtual void DoTimer(long theTimerId);
    virtual void DoUser(long theUserId, void* theData);

    // Coordinate Logical Units:
    virtual CUnits* GetUnits(void) const;
    virtual void SetUnits(CUnits* theCoordinateUnits);

    // Sub-object management:
    virtual const CList* GetSubObjects(void) const = NULL;

protected:
    CUnits* itsUnits; //The logical coordinates of the object

    // Default constructors and cloners
    CBoss(void);
    CBoss(const CBoss& theBoss);
    CBoss& operator=(const CBoss& theBoss);
    virtual ~CBoss(void);

    BOOLEAN IBoss(CGlobalUser *theGlobalUser);

    virtual void UpdateUnits(CUnits* theUnits);

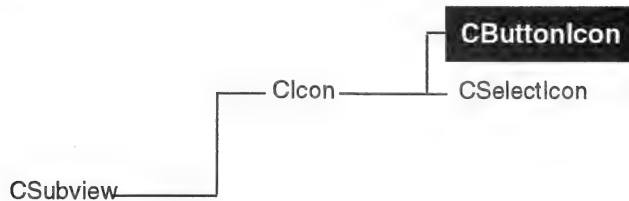
private:
    friend class CApplication; // Allow application to set G
    friend class CUnits; // Allow units to call UpdateUnits

    BOOLEAN IBoss(CGlobalClassLib *theGlobalClassLib);
};

#endif CBoss_H

```

CButtonIcon



Description

CButtonIcon is a class that adds button behavior to an icon. When you click on a button, not only does it produce a command and generate a response, but it also changes faces, becoming indented or inverted. It inverts when the user presses down a mouse button on it and returns to normal when the mouse button is released or when the mouse is moved outside the CButtonIcon object's location. Like all buttons, CButtonIcon objects send a DoCommand message to their enclosures upon receiving a mouse click.

Heritage

Superclass: CIcon

Usage

When you create and initialize each CButtonIcon, it requires three resource IDs for the following icon states:

- enabledRID: the icon is enabled (can receive events)
- disabledRID: the icon is disabled (cannot receive events)
- invertedRID: the icon is inverted

For information on creating these resources, consult the appropriate manual(s) for your platform.

Environment

Under some platforms, the icon drawing appears in the foreground color, as does its title. Open spaces in the drawing appear in the background color. You can set the background and foreground colors. Under other platforms, the icon's color is fixed as defined. For portability, set the colors appropriately even if the information is not used.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

BOOLEAN	itIsHit;	
int	itsPressedRID;	stores the resource ID for a button press

Public Methods

Constructor, Destructor, and Initializer Methods

```
CButtonIcon(CSubview* theEnclosure, const CRect& theRegion);
```

A constructor. theEnclosure is a pointer to the subview that will contain the icon. theRegion is a coordinate location, local to the enclosure, that is used to place the button icon.

```
CButtonIcon(const CButtonIcon& theIcon);
```

A copy constructor that copies all attributes of a button icon. Any views nested within the icon are not copied.

```
CButtonIcon& operator=(const CButtonIcon& theIcon);
```

An assignment operator that copies all attributes of a button icon. Any views nested within the icon are not copied.

```
virtual ~CButtonIcon(void);
```

The destructor. It cleans up after the icon and deletes any views nested within it.

```
BOOLEAN IButtonIcon(int theEnabledRID      = NULLicon,  
                    int theDisabledRID      = NULLicon,  
                    int thePressedRID       = NULLicon,  
                    const CString theTitle  = NULLString,  
                    long theCommand         = NULLcmd,  
                    BOOLEAN isVisible       = TRUE,  
                    GLUETYPE theGlue        = NULLSTICKY);
```

The initializer. Like the initializer of CIcon, it takes a resource ID for the enabled and disabled states of the icon, a visibility state, and a glue type. Also like the CIcon initializer, it takes a theTitle parameter, which is the title of the icon. This title is

displayed underneath the icon and centered. In addition, the CButtonIcon initializer has some button-specific parameters that pertain to what happens when the button is pressed. thePressedRID specifies the resource or the picture that is used to represent the icon in a *pressed* state. theCommand indicates the command that is generated when the button icon is pressed. You can override one or more of these defaults by calling IButtonIcon.

Mouse Events

The following mouse event methods have been overridden to implement button behavior. Mouse events are passed directly to the button icon at the location where the event occurs. Each of the following methods has a theButton argument that specifies which button on a multi-button mouse is used and can have the values 0 (left), 1 (middle), or 2 (right). By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button.

```
virtual void MouseDown(CPoint theLocation,
    short theButton    = 0,
    BOOLEAN isShiftKey = FALSE,
    BOOLEAN isControlKey= FALSE);
```

When the user presses down a mouse button over a button icon containing local coordinate theLocation, the icon receives and handles this event. The icon changes to its inverted state.

```
virtual void MouseMove(CPoint theLocation,
    short theButton    = 0,
    BOOLEAN isShiftKey = FALSE,
    BOOLEAN isControlKey= FALSE);
```

When the user moves the mouse over a button icon containing local coordinate theLocation, the icon receives and handles this event. If the icon is inverted and theLocation is outside its bounds, it becomes uninverted.

```
virtual void MouseUp(CPoint theLocation,
    short theButton    = 0,
    BOOLEAN isShiftKey = FALSE,
    BOOLEAN isControlKey= FALSE);
```

When the user releases a mouse button over a button icon containing local coordinate theLocation, the icon receives and handles this event. If the icon was inverted at this time, a command is sent and the button becomes uninverted.

```
virtual void MouseDouble(CPoint theLocation,
    short theButton    = 0,
    BOOLEAN isShiftKey = FALSE,
    BOOLEAN isControlKey= FALSE);
```

When the user double clicks a mouse button over a button icon containing local coordinate theLocation, the icon receives and handles this event by producing a DoCommand.

Protected Methods

virtual void Press(void);

Takes care of button presses. This method is protected in case you want to derive a class that changes the button behavior.

Private Methods

None.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual void Enable(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
```



```
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey,
                BOOLEAN isControlKey);
virtual void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment&
                           theNewEnvironment);
virtual void SetFont(const FONT& font,
                    BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
```

```

virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;

```

```

virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CIcon

```

virtual void Disable(void);
virtual void Draw(const CRect& theClippingRegion);
virtual void Enable(void);
virtual void Invert(void);
virtual void SetFont(const FONT& theFont, BOOLEAN isUpdate = FALSE);
virtual void SetSizing(BOOLEAN isSizable);
virtual void Size(const CRect& theNewRegion);
virtual void SetTitle(const CString& theTitle);

```

Summary: CButtonIcon.h

```

#ifndef CButtonIcon_H
#define CButtonIcon_H
#include "CIcon.h"

class CButtonIcon : public CIcon
{
public:
    // Construct, Destruct, Initialize:
    CButtonIcon(CSubview* theEnclosure, const CRect& theRegion);
    CButtonIcon(const CButtonIcon& theIcon);
    CButtonIcon& operator=(const CButtonIcon& theIcon);
    virtual ~CButtonIcon(void);

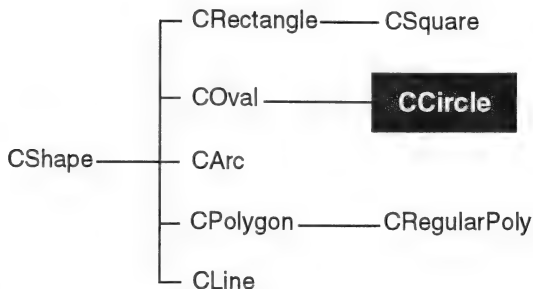
    BOOLEAN IButtonIcon(int theEnabledRID = NULLicon,
        int theDisabledRID = NULLicon,
        int thePressedRID = NULLicon,
        const CString theTitle = NULL,
        long theCommand = NULLcmd,
        BOOLEAN isVisible = TRUE,
        GLUETYPE theGlue = NULLSTICKY);

```

```
// mouse events:
virtual void MouseDouble(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
protected:
    virtual void Press(void);
private:
    BOOLEAN itIsHit;
    int itsPressedRID;
};

#endif CButtonIcon_H
```

CCircle



Description

CCircle objects draw a circle inside a CView object.

Heritage

Superclass: COval

Usage

Create and initialize a CCircle object. Like all CShape classes, this class inherits all properties of CSubview, such as stickiness, enclosure, and so on.

Environment

The border of the circle is drawn with the *pen*, its interior is painted with the *brush*. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width.

Data Members

None.

Public Methods

```
CCircle(CSubview* theEnclosure, const CPoint&
theCenterPoint, UNITS theRadius);
```

A constructor. It takes a pointer to an enclosure, which is the view that contains the CCircle object. In addition, the constructor requires the center point for the circle (a CPoint that is relative to the circle's enclosure) and the radius of the circle.

```
CCircle(const CCircle& theCircle);
```

A copy constructor that creates a new `CCircle` object with the same enclosure, color, visibility attributes, enabled/disabled attributes, environment, and so on as the original `CCircle` object. However, any views nested within the circle are not copied.

```
CCircle& operator=(const CCircle& theCircle);
```

An assignment operator. It copies the attributes of the circle, including the color, glue, environment, visibility state, and so on. However, any views nested within the circle are not copied.

```
virtual ~CCircle(void);
```

The destructor. It cleans up after the `CCircle` object and deletes any views nested in it.

```
BOOLEAN ICircle(BOOLEAN isVisible, Gluetype  
Glue);
```

The initializer, which allows the visibility status of the circle to be set and can take a glue type.

```
virtual void Size(const CRect& aRect);
```

Sizes the circle according to the coordinates of the `theNewSize`. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, `theNewSize`, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

```
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
```

```
virtual void Deactivate(void);
```

```
virtual void Disable(void);
```

```
virtual void DoCommand(long theCommand,void* theData=NULL);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
```

```

virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

void Prepare(const CRect& theClippingRegion);

virtual void PrintDraw(const CRect& theRegion);

virtual void SetCommand(long theCommand);

virtual void SetDoubleCommand(long theCommand);

virtual void SetDragging(BOOLEAN isDraggable);

virtual void SetEnclosure(CSubview* theEnclosure);

virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);

virtual void SetFont(const FONT& font,
BOOLEAN isUpdate = FALSE);

virtual void SetGlue(GLUETYPE theGlue);

virtual void SetId(int theID);

virtual void SetOrigin(const CPoint& theDeltaPoint);

virtual void SetSizing(BOOLEAN isSizable);

virtual void SetTitle(const CString& theNewTitle);

virtual void SetWireFrame(CWireFrame* theNewWireFrame);

virtual void Show(void);

virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);

virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);

virtual void DoActivate(void);

virtual void DoDeactivate(void);

virtual void DoDisable(void);

virtual void DoDraw(void);

virtual void DoDraw(const CRect& theClippingRegion);

virtual void DoEnable(void);

virtual void DoHide(void);

virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);

virtual void DoMouseDown(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

```



```

virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoPrintDraw(const CRect& theClippingRegion);

virtual void DoSetDragging(BOOLEAN isDraggable);

virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);

virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);

virtual void DoSetGlue(GLUETYPE theGlue);

virtual void DoSetOrigin(const CPoint& theDeltaPoint);

virtual void DoSetSizing(BOOLEAN isSizable);

virtual void DoShow(void);

virtual void DoSize(const CRect& theNewSize);

virtual CView* FindDeepSubview(const CPoint& theLocation) const;

virtual FindEventTarget(const CPoint& theLocation) const;

virtual CView* FindSubview(int theViewId) const;

virtual CView* FindSubview(const CPoint& theLocation) const;

virtual void FindSubviews(const CPoint& theLocation,List* theList) const;

CView* GetKeyFocus(void) const;

int GetNumViews(void) const;

virtual CView* GetSelectedView(void) const;

virtual const CList* GetSubObjects(void) const = NULL;

virtual const CList* GetSubviews(void) const;

virtual void PlaceBottomSubview(const CView* theView);

virtual void PlaceTopSubview(const CView* theView);

virtual void RemoveSubview(const CView* theSubview);

virtual void SetKeyFocus(CView *theFocusedView);

virtual void SetSelectedView(CView* theSelectedView);

```

From CShape

None.

From COval

```
virtual void Draw(const CRect& theClippingRegion);
```

Summary: CCircle.h

```

#ifndef CCircle_H
#define CCircle_H
#include "COval.h"

```

```
class CCircle : public COval
{
    public:
        CCircle(CSubview* theEnclosure, const CPoint& theCenterPoint,
            UNITS theRadius);
        CCircle(const CCircle& theCircle);
        CCircle& operator=(const CCircle& theCircle);

        BOOLEAN ICircle(BOOLEAN isVisible, Gluetype Glue);
        virtual void Size(const CRect& aRect);
};

#endif CCircle_H
```

CDesktop



Description

A CDesktop object manages an application's "workspace" or screen window layout. It keeps track of the state of an application's windows—their position, their placement relative to other windows, whether they are iconified, and so on. Every window that is created is put into the desktop.

Heritage

Superclass: none

Usage

One CDesktop object is created per application by the CApplication object. All core XVT-Power++ classes have access to the desktop through the global reference stored by CBoss::GetDesktop. CDesktop is a class that is instantiated internally, but you can derive your own desktop and use it by calling G-> SetDesktop(...).

Public Data Members

None.

Protected Data Members

None.

Private Data Members

CApplication	*itsApplication;	the application to which it belongs
--------------	------------------	-------------------------------------

CWindow	*itsFrontWindow;	the front window
CList	*itsWindowObjects;	list of the windows

Friends

friend class	CSwitchBoard
friend class	CDocument;

Public Methods

Constructor, Destructor, and Initializer Methods

CDesktop(CApplication *theApplication);

A constructor. It takes a pointer to the application to which the desktop belongs (theApplication).

CDesktop(const CDesktop& theDesktop);

A copy constructor. It disables the copying of a desktop.

CDesktop& operator=(const CDesktop& theDesktop);

An assignment operator, which disables the copying of a desktop.

virtual ~CDesktop(void);

The destructor. It cleans up after the desktop but does not destroy any windows.

Desktop Object Methods

virtual void SetFrontWindow(CWindow *theWindow);

Takes a pointer to a window (theWindow) and sets that window to the front of the window stack.

virtual CWindow* GetFrontWindow(void) const;

Returns a pointer to the window that is at the front of the window stack.

virtual int GetNumWindows(void) const;

Returns the number of windows in the desktop.

virtual void PlaceWindow(CWindow *theWindow);

A method that is called to place a window when it is created. You can override this method to meet your application's window placement needs.

```
BOOLEAN FindWindow(CWindow *theWindow) const;
```

Returns a Boolean value of TRUE if theWindow is in the desktop and FALSE if it is not.

Modal Window Utility Method

```
virtual void DoModal(CModalWindow* theWindow);
```

Takes a pointer to a modal window. When DoModal is called, it makes the designated window behave modally. Note that no modal window will behave modally until this method is called. In fact, until DoModal is called, a modal window behaves just like any other window. When DoModal is called, the given modal window takes over the screen, disabling all other windows so that nothing else can happen while it is open. For example, on the Macintosh, when a modal window appears, all other operations are disabled. Most items on the menubar are greyed-out, and all of the windows in the background become disabled. The modal window appears in the middle of the screen, and you cannot even move it or size it until you press the necessary button(s) on the window or respond to it in another way that is indicated. Then everything returns to normal. Modal windows are used when the program needs a certain item of information or a commitment from the user before it can continue. See DoModalWindow.

Protected Methods

Interaction With CDocument

```
virtual void AddWindow(CWindow *theWindow);
```

Adds theWindow to the desktop. Only CDocument objects can access this method, which is called internally.

```
virtual void RemoveWindow(CWindow *theWindow);
```

Removes theWindow from the desktop. Only CDocument objects can access this method, which is called internally.

Interaction With CSwitchBoard

```
void SetActiveWindow(CWindow *theWindow);
```

A method that is called internally when a user clicks on theWindow. It sets this window to an active state, which means that it is the selected window.

Private Methods

None.

Overrides

Summary: CDesktop.h

```
#ifndef CDesktop_H
#define CDesktop_H

#include "PwrNames.h"
#include "xvt.h"
#include "CGlobal.h"
#include "CLists.h"

class CIterator;
class CApplication;
class CWindow;

class CDesktop
{
public:
    // construct & destruct & init
    CDesktop(CApplication *theApplication);
    CDesktop(const CDesktop& theDesktop);
    CDesktop& operator=(const CDesktop& theDesktop);
    virtual ~CDesktop(void);

    // Desktop Object Functions
    virtual void SetFrontWindow(CWindow *theWindow);
    virtual CWindow* GetFrontWindow(void) const;

    virtual int GetNumWindows(void) const;

    virtual void PlaceWindow(CWindow *theWindow);
    BOOLEAN FindWindow(CWindow *theWindow) const;

protected:
    virtual void AddWindow(CWindow *theWindow);
    virtual void RemoveWindow(CWindow *theWindow);

    virtual void SetActiveWindow(CWindow *theWindow);

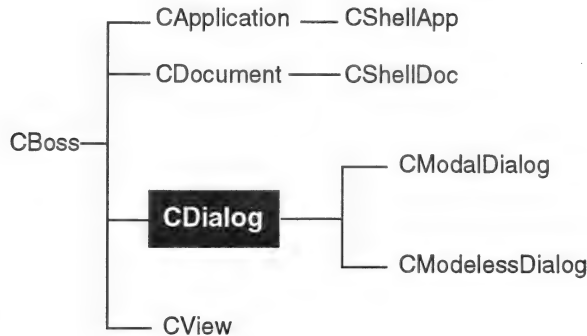
private:
    CApplication *itsApplication; // the application to which it
                                // belongs
    CWindow *itsFrontWindow; // the front window
    CList *itsWindowObjects; // list of the window's

    // Interaction with CDocument:
    friend class CDocument;

    // Interaction with CSwitchBoard:
    friend class CSwitchBoard;
    void SetActiveWindow(CWindow *theWindow);
};

#endif CDesktop_H
```

CDialog



Description

CDialog provides an object that is commonly known as a “dialog box.” We prefer the term “dialog window” because this object is very like a window, with only a few differences. Sometimes, in fact, there is almost no visible difference between a dialog and a window. The main difference in XVT-Power++ is that all of the dialogs created through CDialog are defined in URL resource format.

Dialog windows can be created in two different modes: modal and modeless. When a dialog window is modal, it takes over the screen, disabling all other objects and operations so that nothing else can happen while the dialog window is open. For example, on the Macintosh, when a modal dialog window appears, all other operations are disabled. Most items on the menubar are greyed-out, and all of the windows in the background become disabled. The dialog window appears in the middle of the screen, and you cannot even move it or size it until you press the necessary button(s) on the dialog box. Then everything returns to normal. In short, all objects on the screen except the modal dialog window are frozen, and the dialog window does not go away until the user responds to it. Modal dialog windows are used when the program needs a certain item of information or a commitment from the user before it can continue. Modeless dialogs, on the other hand, do not freeze everything on the screen. They behave like regular windows and thus can go into the background when another window is brought to the front.

Heritage

Superclass: CBoss

Usage

CDialog is an abstract class and thus cannot be instantiated. You must instead instantiate one of its two child classes: CModalDialog or CModelessDialog, which are both defined in this section. First, define a layout for a dialog that contains different controls, such as buttons, fields, text, and so on. Then instantiate the object.

Remember, a dialog window is a resource-defined object. It is derived directly from CBoss and not from CView. While it is very like a view, every object inside it is not defined as a view but rather as a control in a URL resource file. Thus, the dialog window does not act as an enclosure, as do many of the classes derived from CView. You cannot add anything to it; you can only instantiate it and interact with it.

For hands-on practice in creating and using modal dialog windows, consult the “Ask” example that is provided on the XVT-Power++ distribution disk.

Public Data Members

None.

Protected Data Members

CBoss*	itsOwner;	a pointer to any boss that has created the dialog box and is in charge of it. This pointer is used for passing messages and DoCommands back to its owner.
WINDOW	itsXVTWindow;	the window handle of the dialog window
BOOLEAN	itIsEnabled;	a variable indicating whether the dialog box is enabled to receive events

Friends

friend class	CSwitchBoard;	sends events to the dialog object
--------------	---------------	-----------------------------------

Public Methods

Utility Methods

```
virtual void SetTitle(const CString& theTitle);
```


Sets the title of the dialog window. On some platforms, no title is associated with the dialog window. However, you can still set it, and XVT-Power++ will take care of the portability issues.

virtual CString GetTitle(void) const;

Returns the title of the dialog window.

virtual void Enable(void);

Enables the dialog window so that it can receive events.

virtual void Disable(void);

Disables the dialog window so that it cannot receive events.

virtual BOOLEAN IsEnabled(void) const;

Indicates whether the dialog window is enabled to receive events (TRUE) or disabled (FALSE) so that it cannot receive events.

virtual CRect GetFrame(void) const;

Returns the CRect coordinates of the *inside* of the dialog window. These coordinates are relative to the dialog window's border.

virtual CRect GetGlobalFrame() const;

Returns the CRect coordinates of the dialog window's border relative to the entire screen. These dimensions include anything having to do with the outside of the dialog window: border, menubar, size box, and so on.

virtual void Size(const CRect& theNewFrame);

Sizes the dialog window according to the coordinates of theNewSize.

virtual void Close(void);

Closes the dialog window.

virtual WINDOW GetXVTWindow(void) const;

Returns the dialog window's XVT Portability Toolkit window handle. This method is useful if you want to manipulate the dialog window at the XVT Portability Toolkit level.

virtual WINDOW FindXVTControl(int theControlId);

Given a control ID, returns a window handle to that control. Dialog windows, which are defined as resources, may contain

other controls, each of which has a control ID as defined in URL.

Nested Control Handlers

CDialog is a “wrapper” class. It behaves as a wrapper to the XVT Portability Toolkit dialog functionality. As a wrapper, CDialog divides up all the events that can come to the dialog window and sends messages to different control handlers, which are listed and described in this subsection. Any time an event comes to a dialog window, one of these control handlers is called. For example, CDialog calls DoButton when it receives an XVT Portability Toolkit button event. Each of the following control handlers is a hook. You must write the code required by the XVT Portability Toolkit to handle each type of control event.

```
virtual void DoButton(int theControlId);
```

A method that is called when a dialog window receives a button event. It takes the control ID of the button. You must override DoButton, putting in a switch statement indicating which button generated the event.

```
virtual void DoRadioButton(int theControlId);
```

A method that is called when a dialog window receives a radio button event. It takes the control ID of the radio button. You must override DoRadioButton, putting in the mechanism for deselecting one radio button and selecting another.

```
virtual void DoCheckBox(int theControlId);
```

A method that you must override to call the XVT Portability Toolkit's win_check_box function.

```
virtual void DoEdit(int theControlId, BOOLEAN  
isFocusChanged, BOOLEAN isActive);
```

A method that is called when a dialog window receives an edit control event. It does not give a character, so you must get the contents of the edit box using the XVT Portability Toolkit interface to the edit controls. theControlId is the URL resource ID number of the edit control. isFocusChanged takes a Boolean value of TRUE if the focus has changed. If this value is TRUE, you can check the isActive parameter to see if the focus change is to be active or inactive for the particular edit box inside the dialog window.

```
virtual void DoListBox(int theControlId,  
BOOLEAN isDoubleClick);
```

A method that is called when the user clicks on a list box inside a dialog window. `theControlId` is the URL resource ID number of the list box control. If the click is a double click, then the Boolean `isDoubleClick` is set to TRUE.

```
virtual void DoHScroll(int theControlId,  
    SCROLL_CONTROL theEvent, short thePosition);
```

A method that is called when a horizontal scrollbar event occurs inside a dialog window. `theControlId` is the URL resource ID number of the scrollbar control. `theEvent` takes an XVT Portability Toolkit value, and `thePosition` is the position of the scrollbar's thumb.

```
virtual void DoVScroll(int theControlId,  
    SCROLL_CONTROL theEvent, short thePosition);
```

A method that is called when a vertical scrollbar event occurs inside a dialog window. `theControlId` is the URL resource ID number of the scrollbar control. `theEvent` takes an XVT Portability Toolkit value, and `thePosition` is the position of the scrollbar's thumb.

```
virtual void DoListButton(int theControlId);
```

A method that is called when a list button is pressed. `theControlId` is the URL resource ID number of the list button. As described in the *XVT Programmer's Guide*, a list button is a button that invokes a list box displaying a list of options when the user presses it. A list button has a title beside it, which is the current option selected from the list. If the list is empty, no title appears beside the button.

```
virtual void DoListEdit(int theControlId,  
    BOOLEAN isFocusChanged, BOOLEAN isActive);
```

A method that is called when a dialog box receives a list edit event. `theControlId` is the URL resource ID number of the list edit. As described in the *XVT Programmer's Guide*, a list edit is an edit control field that invokes a pop-up list of alternate shorthand text input items, in other words, alternate pieces of text that can replace the text contained in the edit field. This list appears only when the user presses the part of the list edit control that displays the list component. Like `DoEdit`, `DoListEdit` has an `isFocusChanged` parameter that takes a Boolean value of TRUE if the focus has changed. If this value is TRUE, you can check the `isActive` parameter to see if the focus change is to be active or inactive for the particular edit box inside the dialog window.

```
virtual void DoKey(int theChar, BOOLEAN
    isShift, BOOLEAN isControl);
```

A method that is called when a dialog box receives a keyboard event that is not absorbed by one of the controls it contains, such as an edit control. `theChar` is the key that was pressed. `isShift` and `isControl` indicate whether the SHIFT key or CONTROL key was pressed in conjunction with `theChar`.

Inherited Methods

```
virtual const CList* GetSubObjects(void) const;
```

A method that belongs to all classes inheriting from `CBoss`. Here `GetSubObjects` returns a value of `NULL` because dialog boxes do not have a list of XVT-Power++ subobjects; they just have a list of controls, which are not XVT-Power++ objects.

Protected Methods

Methods Called Only by the Dialog Event Handler (Thus Protected)

```
CDialog(CBoss* theOwner, WIN_TYPE
    theXVTWindowType, int theResourceId,
    EVENT_MASK theXVTMask = EM_ALL);
```

A constructor. It is protected because `CDialog` is an abstract class and thus cannot be instantiated directly. It takes a pointer to the owner of the dialog box. `theXVTWindowType` takes an XVT Portability Toolkit window type, which indicates whether the dialog window is modal or modeless. `theResourceId` is the URL resource ID number of the dialog window, and `theXVTMask` allows you to mask any types of events that you want the dialog window to receive. The XVT Portability Toolkit's event mask constants allow you to restrict the events that can be sent to the event handler for a window or dialog. These constants can be OR'd together. As noted in the *XVT Programmer's Guide*, the event mask is normally set to `EM_ALL` for no restriction. A value of `EM_NONE` causes no events to be sent to the dialog window.

```
virtual ~CDialog();
```

The destructor. It is called during the `E_DESTROY` event.

```
virtual void DoCreate(void);
```

Once the dialog window is completely instantiated, this method initializes some of the data structures. You can override it to add to its functionality.

```
virtual void DoActivate(void);
```

Activates a dialog window.

```
virtual void DoDeactivate(void);
```

Deactivates a dialog window. If a dialog window is modal, it normally never receives a deactivate message—unless another dialog box is invoked.

```
virtual void SizeWindow(int theWidth, int  
theHeight);
```

Sizes a dialog window according to the given width and height.

```
virtual void DoControl(int theControlId,  
CONTROL_INFO theInfo);
```

The dispatcher of all control events. `theControlId` is the URL resource number of the control. `theInfo` is the type of XVT Portability Toolkit event generated by the control. `DoControl` translates the given event into one of the “Do-” methods described in the section on “Nested Control Handlers”.

```
virtual void DoTimer(long theTimerId);
```

A method that is called when the XVT Portability Toolkit generates a timer (E_TIMER) event. You set a timer using the XVT Portability Toolkit’s `set_timer` function, which takes a window and a time interval (in milliseconds) and returns an ID number for the timer. This is the ID number that is passed to `theTimerId` of `DoTimer`. When the timer generates a timer event, `CSwitchBoard` passes it to the XVT Portability Toolkit-designated window via `DoTimer`. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

```
virtual void DoUser(long theUserId, void*  
theData);
```

A method that is called when the XVT Portability Toolkit generates a user (E_USER) event. As noted in the *XVT Programmer’s Guide*, user events allow you to pass custom events to windows and dialogs. `theUserId` is the ID number that designates a particular kind of user event. The void pointer `theData` can be used to pass any user-defined structure to `DoUser`, just as you would for a `DoCommand`. As with timer events, `CSwitchBoard` passes a user event to a designated window. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

Classes Derived From CDialog: CModalDialog

Description

See the description of CDialog at the beginning of this section. It explains what a modal dialog window is.

Public Methods

```
CModalDialog(CBoss* theOwner, int  
theResourceId, EVENT_MASK theXVTMask =  
EM_ALL);
```

The constructor. Like the inherited constructor, it takes a pointer to the owner, which is the CBoss object that instantiates it. The pointer is used to pass messages and DoCommands to the dialog's owner. theResourceId is the URL resource ID number of the modal dialog. theXVTMask allows you to mask any types of events that you want the dialog window to receive. The XVT Portability Toolkit's event mask constants allow you to restrict the events that can be sent to the event handler for a window or dialog. These constants can be OR'd together. As noted in the *XVT Programmer's Guide*, the event mask is normally set to EM_ALL for no restriction. A value of EM_NONE causes no events to be sent to the dialog window.

Classes Derived From CDialog: CModelessDialog

Description

See the description of CDialog at the beginning of this section. It explains what a modeless dialog window is.

Public Methods

```
CModelessDialog(CBoss* theOwner, int  
theResourceId, EVENT_MASK theXVTMask =  
EM_ALL);
```

A constructor. It initializes a modeless dialog window. Like the inherited constructor, it takes a pointer to the owner, which is the CBoss object that instantiates it. The pointer is used to pass messages and DoCommands to the dialog's owner. theResourceId is the URL resource ID number of the modal dialog. theXVTMask allows you to mask any types of events that you want the dialog window to receive. The XVT Portability Toolkit's event mask constants allow you to restrict the events that can be sent to the event handler for a window or dialog. These constants can be OR'd together. As noted in the *XVT Programmer's Guide*, the event mask is normally set to EM_ALL

for no restriction. A value of EM_NONE causes no events to be sent to the dialog window.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoUser(long theUserId, void* theData);
virtual const CList* GetSubObjects(void) const;
virtual CUnits* GetUnits(void) const;
virtual void SetUnits(CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

Summary: CDialog.h

```
#ifndef CDialog_H
#define CDialog_H

#include "PwrNames.h"
#include "CBoss.h"

class CRect;
class CList;

class CDialog : public CBoss
{
public:
    // Dialog utility:

    virtual void SetTitle(const CString& theTitle);
    virtual CString GetTitle(void) const;

    virtual void Enable(void);
    virtual void Disable(void);
    virtual BOOLEAN IsEnabled(void) const;

    virtual CRect GetFrame(void) const;
    virtual CRect GetGlobalFrame() const;

    virtual void Size(const CRect& theNewFrame);

    virtual void Close(void);

    virtual WINDOW GetXVTWindow(void) const;

    virtual WINDOW FindXVTControl(int theControlId);
```

```

        // Nested Control Handlers:
        virtual void DoButton(int theControlId);
        virtual void DoRadioButton(int theControlId);
        virtual void DoCheckBox(int theControlId);
        virtual void DoEdit(int theControlId,
                            BOOLEAN isFocusChanged, BOOLEAN isActive);
        virtual void DoListBox(int theControlId, BOOLEAN isDoubleClick);
        virtual void DoHScroll(int theControlId,
                               SCROLL_CONTROL theEvent, short thePosition);
        virtual void DoVScroll(int theControlId,
                               SCROLL_CONTROL theEvent, short thePosition);
        virtual void DoListButton(int theControlId);
        virtual void DoListEdit(int theControlId,
                                BOOLEAN isFocusChanged, BOOLEAN isActive);
        virtual void DoKey(char theChar, BOOLEAN isShift,
                           BOOLEAN isControl);

        // Inherited:
        virtual const CList* GetSubObjects(void) const;

protected:
        CBoss* itsOwner;
        WINDOW itsXVTWindow;
        BOOLEAN itIsEnabled;

        friend class CSwitchBoard;

        // Methods called only by the dialog event handler
        // (thus protected):
        CDialog(CBoss* theOwner,
                WIN_TYPE theXVTWindowType,
                int theResourceId,
                EVENT_MASK theXVTMask = EM_ALL);

        virtual ~CDialog();

        virtual void DoCreate(void);
        virtual void DoActivate(void);
        virtual void DoDeactivate(void);

        virtual void SizeWindow(int theWidth, int theHeight);
        virtual void DoControl(int theControlId, CONTROL_INFO theInfo);
        virtual void DoTimer(long theTimerId);
        virtual void DoUser(long theUserId, void* theData);
};

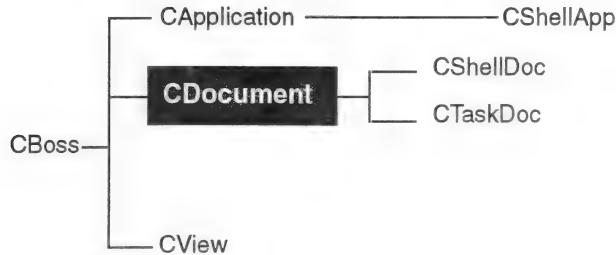
class CModalDialog : public CDialog
{
public:
        CModalDialog(CBoss* theOwner,
                    int theResourceId,
                    EVENT_MASK theXVTMask = EM_ALL);

```



```
};  
class CModelessDialog : public CDialog  
{  
public:  
    CModelessDialog(CBoss* theOwner,  
                    int theResourceId,  
                    EVENT_MASK theXVTMask = EM_ALL);  
};  
#endif CDialog_H
```

CDocument



Description

The `CDocument` class resides at the second layer of the XVT-Power++ application framework. Like `CApplication`, it has access to the functions on the menubar: Open, Close, Print, Save, and so on. `CDocument` objects create and manage a common set of windows:

- Providing access to the data to be displayed inside views, in the form of files, records, hooks to a database, and so on.
- Functioning as a central means of communication for changes and updates in different windows.

A `CDocument` object can display the same data in one or more windows—in both a spreadsheet and a graph, for example. A document maintains any data shared by its windows. Finally, a particular `CDocument` may handle events, such as menu commands, that one of its windows has received.

Heritage

Superclass: `CBoss`

Subclass: `CShellDoc`

Usage

This is an abstract class from which application-specific documents derive their properties. The user-defined `CApplication`-derived class has the responsibility to instantiate one or more user-defined and derived `CDocument` objects.

Group your application windows by functionality or other similarities and use one `CDocument` object to create and manage

those windows. To derive a class from CDocument, overload the BuildWindow method, which is a pure virtual. Put into this method the code to create the specific type of view for the data. Typically, you will also overload the constructor so that it calls the BuildWindow method and starts the window. Also, DoCommand is a very good method to overload. For an example of how to overload the methods on CDocument, see CShellDoc. This example class overrides the BuildWindow method, the constructor, and the DoCommand.

Public Data Members

None.

Protected Data Members

Clist	*itsWindows;	list of the document's windows
FILE_SPEC*	itsXVTFilePointer;	pointer to document file

Private Data Members

CEnvironment	*itsEnvironment;	contains the environment (colors, fonts, pen width, and so on)
int	itsId;	the document ID assigned by the application
CApplication	*itsApplication;	its creator
BOOLEAN	itIsSaved;	whether the data needs to be saved

Friends

friend class	CWindow;	allows windows to self-add and self-remove
--------------	----------	--

Public Methods

Constructor and Destructor Methods

```
CDocument(CApplication* theApplication, int theId);
```

A constructor. The application object instantiates each document and is responsible for managing the set of document objects. It keeps track of the documents by means of an associated list of document ID numbers. Each document has a unique ID number that is entered onto this list when the

document is instantiated. By default, a CDocument object uses the CApplication object's environment, so its environment is set to NULL.

Each CDocument object is responsible for managing a set of one or more windows for viewing its data; it has an associated list of windows called `itsWindows`. Initially, this list is empty.

The data member `itIsSaved` is initially set to FALSE, meaning that there is no data to save.

CDocument(const CDocument& theDocument);

A constructor that is used for copying a document object, `theDocument`. It does not copy the document's windows; it just clones the actual object.

CDocument& operator=(const CDocument& theDocument);

An assignment operator that is used for the copy operation. It takes a constant CDocument object. It does not copy a document's windows.

virtual ~CDocument(void);

The destructor. It is responsible for cleaning up and removing the document from the application at shutdown. It closes all of a document's windows, deletes the document's environment, and deletes its list of windows.

CDocument Utilities

BOOLEAN NeedsSaving(void) const;

Returns a Boolean value indicating whether the document's data needs to be saved. Every time the data is modified, the Boolean data member `itIsSaved` is set to FALSE, meaning that the data needs to be saved. `NeedsSaving` returns the current state of the document as indicated by the value of the `itIsSaved` data member.

void SetSave(BOOLEAN isSaved);

Sets the "NeedsSaving" state of the document, thus triggering a call to `UpdateMenus`, which enables "Save" on the File menu.

int GetId(void) const;

Returns the ID number of the document.

virtual BOOLEAN IsEnvironmentShared(void) const;

A convenient method for finding out if a view is sharing an environment with another object. It returns FALSE if the view is using an environment of its own.

```
virtual const CEnvironment*  
GetEnvironment(void) const;
```

Returns the environment of the document via a constant CEnvironment pointer. If no environment is specified for the document, GetEnvironment returns the application's environment.

```
virtual void SetEnvironment(const CEnvironment&  
theNewEnvironment, BOOLEAN isUpdate =  
FALSE);
```

Sets the environment for a document, using theNewEnvironment that is passed to it. As soon as you use SetEnvironment to give a document an environment of its own, the document uses that environment instead of a shared environment.

When the global environment is changed, all documents sharing that environment get a SetEnvironment message with the isUpdate parameter set to TRUE. The isUpdate parameter indicates whether this SetEnvironment message is simply an update message or whether it is a "real" SetEnvironment message meaning that this particular document should set its own environment to the new environment.

```
virtual void DoSetEnvironment(const  
CEnvironment& theEnvironment, BOOLEAN  
isUpdate = FALSE);
```

Sets the document's environment to the given environment. The document in turn notifies each of its windows of the environment parameters of theNewEnvironment. The isUpdate parameter indicates whether this DoSetEnvironment message is simply an update message or whether it is a "real" DoSetEnvironment message meaning that this particular document should set its own environment to the new environment and pass this change to all of its windows who are sharing its environment.

```
virtual void ChangeFont(FONT theFont, FONT_PART  
thePart);
```

If the document has an environment, this method changes the font of that environment to theFont. thePart indicates which part of the font actually changed because of a Font menu selection (family, style, or size). If the document does not have

an environment of its own, the event is propagated upwards to the application. This method is called after a Font menu selection. To change the font of the document internally, call `SetEnvironment`.

```
virtual PRINT_RCD* GetPrintRecord(void) const;
```

Returns the application's global print record. It is up to you to add document-specific print records that can be saved with documents so that each document prints differently. Then you can override `GetPrintRecord` within a user-specific document class to return whatever print record is being used for that document.

Window Methods

```
virtual void BuildWindow(void) = NULL;
```

This pure virtual function must be overridden. This is where you put the code that instantiates a view. The `BuildWindow` mechanism can be called from `DoOpen` or `DoNew`. It is where we recommend putting code for creating windows.

```
virtual void CloseAll(void);
```

Goes through the `CDocument` object's list of windows, notifying each one to close. This is a way of deleting all of a document's views. This method disregards whether an data must be saved.

```
virtual CWindow* FindWindow(int theId) const;
```

Given an ID number to a view, it returns a pointer to the window.

```
int GetNumWindows(void);
```

Returns the number of windows on the `CDocument` object's list of windows.

```
const CList* GetWindows(void) const;
```

Returns a list of the document's windows.

Event and Application Command Methods

```
virtual void DoCommand(long theCommand, void* theData=NULL);
```

The document-level `DoCommand` handler. It should be overridden and called by default to pass the `DoCommand` up to the application. This `DoCommand` is not associated with any XVT Portability Toolkit event, although it can be generated from an XVT Portability Toolkit event. The void pointer `theData` can be

used to pass any user-defined structure to the DoCommand method.

```
virtual void DoMenuCommand(MENU_TAG  
    theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
    isControlKey);
```

This method is very similar to the DoMenuCommand of CApplication. It is called when the user makes a selection from the menubar. The menu event can come directly from a window to the document. theMenuItem is the menu item number as defined in the URL menubar definition. isShiftKey and isControlKey specify whether the mouse is used with the SHIFT key or CONTROL key.

DoMenuCommand has some predefined actions it takes upon the File menu, which are just like those of CApplication. The methods called by these predefined functions are presented immediately after this description of DoMenuCommand. All of the methods called by these defaults call the default mechanism, which in turn calls the CApplication object. When you override a method, always be sure to call the inherited method for these objects.

M_FILE_CLOSE

Calls the default DoClose method when the user selects "Close" from the menu. See DoClose.

M_FILE_SAVE

Calls the default DoSave method when the user selects "Save" from the menu. See DoSave.

M_FILE_SAVE_AS

Calls the default DoSaveAs method when the user selects "Save" **As** from the menu. See DoSaveAs.

M_FILE_PG_SETUP

Calls the default DoPageSetup method when the user selects "Page Setup" off the menu. See DoPageSetup.

M_FILE_PRINT

Calls the default DoPrint method when the user selects "Print" off the menu. See DoPrint.

M_FILE_OPEN

Calls the default DoOpen method when the user selects "Open" off the menu. See DoOpen.

M_FILE_NEW

Calls the default DoNew method when the user selects "New" off the menu. See DoNew.

File Menu Item Events

virtual BOOLEAN DoClose(void);

Closes all of a document's windows and returns TRUE if all the windows closed successfully. If the document needs saving, a dialog box prompts the user to save, cancel, or discard. For details on the usage of this method, see the Tutorial in the *XVT-Power++ Guide*.

Override:

Specify how to close all the documents in your application, perhaps prompting the user to save certain information before closing.

virtual BOOLEAN DoPageSetUp(void);

Passes on the page set-up task to the application. You can override this method in order to provide document-specific page set-up information.

virtual BOOLEAN DoPrint(void);

Goes through the document's list of windows, dumps each window onto a separate page, and sends the results to the printer. Currently, DoPrint just does a bitmap dump of the windows. You may want to override this method to do some further layout.

virtual BOOLEAN DoOpen(void);

Calls BuildWindow after prompting the user, via a file dialog, for a file name. For details on the use of this method, see the Tutorial in the *XVT-Power++ Guide*.

virtual BOOLEAN DoSave(void);

Saves the document's data and returns a Boolean value of TRUE if saving was successful; you must provide the appropriate action(s) in your derived class by overriding this method. This method automatically sets the state of the document's NeedsSaving flag.

virtual BOOLEAN DoSaveAs(void);

Saves the document's data and returns a Boolean value of TRUE if saving was successful. This method prompts the user for a file name using a file name dialog box. It then calls DoSave to actually save the document. For details on the usage of this method, see the Tutorial in the *XVT-Power++ Guide*.

virtual BOOLEAN DoNew(void);

Creates a new document when there is no data, unlike `Open` which gives access to existing data. `BuildWindow` is called if the operation succeeds in creating a window.

virtual void UpdateMenus(void);

Provides the means for the document to set up the appropriate menus—checking menu items, unchecking them, and so forth—based on the document's state.

This method is also called when a window is selected. The window notifies its associated document that it can update its menu, if necessary. By default, the document object updates the save and deactivates the save, depending on whether the `itIsSaved` flag has been set. If the data needs to be saved, the File menubar "Save" is enabled. If it does not need to be saved, the menu item is disabled. It is important to call the inherited if you override `UpdateMenus` because it does some operations for you automatically, such as the saving functions. When a window object updates its menus, the update event can pass up to its parent and chain upward, stopping at any point, till it reaches the `CApplication` object.

virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);

A method that is called when there is a keyboard event. When the user presses a key on the keyboard, the XVT Portability Toolkit sends an `E_CHAR` event to the switchboard, which passes this event to the selected window. The window checks to see if one of its subviews is selected, and, if so, passes the event to the appropriate subview. If no subview is selected, the keyboard event propagates upwards from the window to the document and finally to the application. A keyboard event goes directly to the application if no windows are open.

`theKey` is the ASCII number of the character key that was pressed; `isShiftKey` and `isControlKey` indicate whether the SHIFT key or CONTROL key was pressed in conjunction with the character key. You can override `DoKey` to do something specific to a keyboard action within the application. The default mechanism does nothing.

virtual const CList* GetSubObjects(void) const;

A pure virtual method that must be defined by any class derived from `CBoss`, specifically, `CApplication`, `CDocument`, `CView`, and `CSubview`. It returns a list of any subobjects associated with a given object. For example, the application would return a list

of all its documents, a document would return a list of all its windows, a window would return a list of all its enclosed views, a subview would return a list of all its enclosed views, and so on.

Event Handler Methods

```
virtual void DoTimer(long theTimerId);
```

A method that is called when the XVT Portability Toolkit generates a timer (E_TIMER) event. You set a timer using the XVT Portability Toolkit's `set_timer` function, which takes a window and a time interval (in milliseconds) and returns an ID number for the timer. This is the ID number that is passed to `theTimerId` of `DoTimer`. When the timer generates a timer event, `CSwitchBoard` passes it to the XVT Portability Toolkit-designated window via `DoTimer`. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

```
virtual void DoUser(long theUserId, void* theData);
```

A method that is called when the XVT Portability Toolkit generates a user (E_USER) event. As noted in the *XVT Programmer's Guide*, user events allow you to pass custom events to windows and dialogs. `theUserId` is the ID number that designates a particular kind of user event. The void pointer `theData` can be used to pass any user-defined structure to `DoUser`, just as you would for a `DoCommand`. As with timer events, `CSwitchBoard` passes a user event to a designated window. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

Protected Methods

```
BOOLEAN IDocument();
```

The initializer. This method initializes internal document structures. Your derived `CDocument` initializer should call this method.

CDocument Utilities

```
virtual void SetId(int theId);
```

Sets the document's ID number.

```
virtual void RemoveWindow(CWindow *theWindow);
```

Given a window, removes the pointer to this window from the document's list of windows. It cannot remove a NULL window.

```
virtual void AddWindow(CWindow * theWindow);
```

Given a window, adds a pointer to this window to the document's list of windows. If the window is deleted, then the pointer on the list is automatically removed.

Methods Inherited But Not Overridden

From CBoss

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

```
virtual void UpdateUnits(CUnits* theUnits);
```

Overrides

You must override BuildWindow in order to attach windows to the document.

Summary: CDocument.h

```
#ifndef CDocument_H
#define CDocument_H

#include "CBoss.h"

class CList;
class CDesktop;
class CApplication;
class CWindow;
class CEnvironment;

class CDocument : public CBoss
{
public:
    // construct, destruct, and init
    CDocument(CApplication* theApplication, int theId);
    CDocument(const CDocument& theDocument);
    CDocument& operator=(const CDocument& theDocument);
    virtual ~CDocument(void);

    // utility functions
    BOOLEAN NeedSaving(void) const ;
    void SetSave(BOOLEAN isSaved);

    int GetId(void) const;

    virtual BOOLEAN IsEnvironmentShared(void) const;
    virtual const CEnvironment* GetEnvironment(void) const;
    virtual void SetEnvironment(const CEnvironment& theEnvironment,
                                BOOLEAN isUpdate = FALSE);
    virtual void DoSetEnvironment(const CEnvironment& theEnvironment);
};
```

```

virtual void ChangeFont(FONT theFont, FONT_PART thePart);
virtual PRINT_RCD* GetPrintRecord(void) const;

// window functions
virtual void BuildWindow(void) = NULL;
virtual void CloseAll(void);
virtual CWindow* FindWindow(int theId) const;
int GetNumWindows(void);

// event and application command functions
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey,
                           BOOLEAN isControlKey);

virtual BOOLEAN DoClose(void);
virtual BOOLEAN DoPageSetUp(void);
virtual BOOLEAN DoPrint(void);
virtual BOOLEAN DoOpen(void);
virtual BOOLEAN DoSave(void);
virtual BOOLEAN DoSaveAs(void);
virtual BOOLEAN DoNew(void);

virtual void UpdateMenus(void);

virtual void DoKey(int theKey, BOOLEAN isShiftKey,
                  BOOLEAN isControlKey);

// Other event handler methods:
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);

protected:
    CList *itsWindows; // list of the document's windows
    FILE_SPEC* itsXVTFilePointer; //pointer to document file

    BOOLEAN IDocument();

    // CDocument utilities
    virtual void SetId(int theId);

    virtual void RemoveWindow(CWindow *theWindow);
    virtual void AddWindow(CWindow * theWindow);

private:
    friend class CWindow; // Allow windows to self-add and self-remove
    CEnvironment *itsEnvironment; // contains env
    int itsId; // the document's id assigned by application
    Application *itsApplication; // its creator
    BOOLEAN itIsSaved; // does the data need to be saved

};

#endif CDocument_H

```

CEnvironment

CMem	CGlue	CSwitchBoard
CError	CDesktop	CStartup
CEnvironment		CGlobalClassLib
		CGlobalUser
	CGlobal	CResourceMgr
		CPrintMgr

Description

CEnvironment is a utility class that contains environment information for objects in the XVT-Power++ class hierarchy, such as:

- font type
- background and foreground colors
- line width
- brush pattern
- drawing mode
- anything that has to do with painting or displaying something on the screen that can have different attributes
- color/monochrome; several methods for getting and setting attributes

By default, there is a global environment object that is shared by every displayable object in an XVT-Power++ application. An environment object can be attached at any level in XVT-Power++, all the way from the application object to the deepest subview. An environment propagates downward, so many objects can share the same environment.

Many of the environment settings take the XVT Portability Toolkit values, as shown in the following tables. You are not limited to the colors listed in the color table. For details, see the *XVT Programmer's Guide*.

M_COPY	Ignores what is on the screen and copies drawn pixels. All XVT Portability Toolkit implementations obey this rule at all times. Use only this mode when printing because some print drivers cannot handle any other method of transferring pixels to paper.
M_XOR	Temporarily shows something on the screen so that the screen can be restored by drawing the same shape again, rather than by updating the screen with what was there before. What was previously on the screen is probably irrelevant




Table 1. Common XVT Portability Toolkit Drawing Modes

normal font	small_font
big font	fixed_font

Table 2. Predefined XVT Portability Toolkit Font Variables

Color	Color Symbol
#define COLOR_RED	0x01FF0000L
#define COLOR_GREEN	0x0200FF00L
#define COLOR_BLUE	0x030000FFL
#define COLOR_CYAN	0x0400FFFFL
#define COLOR_MAGENTA	0x05FF00FFL
#define COLOR_YELLOW	0x06FFFF00L
#define COLOR_BLACK	0x07000000L
#define COLOR_DKGRAY	0x08404040L
#define COLOR_GRAY	0x09808080L
#define COLOR_LTGRAY	0x0AC0C0C0L
#define COLOR_WHITE	0x0BFFFFFFL

Table 3. Predefined XVT Portability Toolkit Color Variables

PAT_SOLID	Solid pattern	
PAT_HORZ	Horizontal	
PAT_VERT	Vertical	





PAT_FDIAG	Forward diagonal	
PAT_BDIAG	Backward diagonal	
PATCROSS	Crisscross	
PAT_DIAGCROS	Diagonal crisscross	

Table 4. Predefined XVT Portability Toolkit Brush Patterns

PAT_SOLID	A normal pen that draws a solid line in the specified width with the specified color
PAT_HOLLOW	No pen at all; shapes with a brush do not have a border around them, and shapes with only a pen are not seen at all.
PAT_RUBBER	Used for rubberbanding; a grayish or dotted line that allows a user to stretch a rectangle or other shape by moving the mouse.

Table 5. Predefined XVT Portability Toolkit Pen Patterns

Heritage

Superclass: none

Usage

A global environment object is automatically created for the application. In addition, you can create CEnvironment objects for other individual objects that should have different “environmental” attributes. Consult the section in this manual on each of the view classes to see how a class uses the environment.

Private Data Members

FONT	itsFont;	the system wide font (see Table 2)
COLOR	itsCBackgroundColor;	color background (see Table 3)
COLOR	itsCForegroundColor;	color foreground (see Table 3)
COLOR	itsBBackgroundColor;	black and white background (see Table 3)
COLOR	itsBForegroundColor;	black and white

		foreground (see Table 3)
COLOR	itsBrushColor;	brush color (see Table 3)
COLOR	itsPenColor;	pen color (see Table 3)
COLOR	itsMonoPenColor;	pen monochrome color
COLOR	itsMonoBrushColor;	brush monochrome color
PAT_STYLE	itsBrushPattern;	the brush pattern (see Table 4)
short	itsPenWidth;	the pen width
PAT_STYLE	itsPenPattern;	the pen pattern (see Table 5)
DRAW_MODE	itsDrawingMode;	the drawing mode (see Table 1)
static BOOLEAN	itIsColor;	whether DISPLAY is in color

Public Methods

Constructor, Destructor, and Initializer Methods

```

CEnvironment(COLOR theBackground    = COLOR_WHITE,
             COLOR theForeground    = COLOR_BLACK,
             COLOR theBrushColor    = COLOR_WHITE,
             PAT_STYLE theBrushPattern = PAT_SOLID,
             COLOR thePenColor      = COLOR_BLACK,
             PAT_STYLE thePenPattern = PAT_SOLID,
             short thePenWidth      = 1,
             const FONT& theFont    = STDFONT,
             DRAW_MODE theDrawingMode = M_COPY);

```

A constructor. `theBackground` is the background color of the object; `theForeground` is the color of the object's foreground. `theBrushColor` is the color of the interior fill for closed objects; `theBrushPattern` is the pattern style of the interior fill. The "pen" is used to draw lines, which have a color (`thePenColor`), a pattern (`thePenPattern`), and a width (`thePenWidth`) measured in pixels. This constructor also specifies the object's font type (`theFont`) and drawing mode. For more information, see Tables 1 through 5.

```
virtual ~CEnvironment(void);
```

The destructor, which cleans up after the environment object.


```

BOOLEAN IEnvironment(COLOR theBackground      = COLOR_WHITE,
                      COLOR theForeground      = COLOR_BLACK,
                      COLOR theBrushColor      = COLOR_WHITE,
                      PAT_STYLE theBrushPattern= PAT_SOLID,
                      COLOR thePenColor       = COLOR_BLACK,
                      PAT_STYLE thePenPattern  = PAT_SOLID,
                      short thePenWidth       = 1,
                      const FONT& theFont     = STDFONT,
                      DRAW_MODE theDrawingMode = M_COPY);

```

The initializer. theBackground is the background color of the object; theForeground is the color of the object's foreground. theBrushColor is the color of the interior fill for closed objects; theBrushPattern is the pattern style of the interior fill. The "pen" is used to draw lines, which have a color (thePenColor), a pattern (thePenPattern), and a width (thePenWidth) in pixels. This constructor also specifies the object's font type (theFont) and drawing mode. For more information, see Tables 1 through 5.

Color Utility Methods

```

virtual void SetColor(COLOR theBackground      = COLOR_WHITE,
                      COLOR theForeground      = COLOR_BLACK,
                      COLOR theBrushColor      = COLOR_WHITE,
                      COLOR thePenColor       = COLOR_BLACK,
                      COLOR theOffBackground  = COLOR_WHITE,
                      COLOR theOffForeground  = COLOR_BLACK,
                      COLOR theOffBrushColor  = COLOR_WHITE,
                      COLOR theOffPenColor    = COLOR_BLACK);

```

Sets the colors of an object: its background and foreground colors, the interior fill of closed shapes (theBrushColor), and the color of lines (thePenColor). theOffBackground and theOffForeground specify the default colors when color has been turned off and monochrome is being used. Similarly, theOffBrushColor and theOffPenColor specify the default brush and pen colors when monochrome is being used.

```
COLOR GetForegroundColor(void) const;
```

Returns the object's foreground color.

```
virtual void SetForegroundColor(COLOR theColor,
                                COLOR theOffColor = COLOR_BLACK);
```

Given a color, sets the object's foreground color. See Table 3. theOffColor specifies the default color of the foreground when monochrome is being used.

```
COLOR GetBackgroundColor(void) const;
```

Returns the object's background color.

```
virtual void SetBackgroundColor(COLOR theColor,  
    COLOR theOffColor = COLOR_WHITE);
```

Given a color, sets the object's background color. See Table 3. theOffColor specifies the default color of the background when monochrome is being used.

Font Methods

```
FONT* GetFont(void) const;
```

Returns the type of the font that has been set for the object.

```
virtual void SetFont(const FONT& theFont,  
    BOOLEAN isUpdate = FALSE);
```

Given a reference to a font (theFont), sets the font for the object. See Table 2. The isUpdate parameter indicates whether this SetFont message is simply an update message or whether it is a "real" SetFont message meaning that this particular view should set its own font to the new font.

Pen Methods

A pen is used to draw lines. The following methods allow you to set and get the pen properties of an object.

```
virtual void SetPen(COLOR theColor, short  
    thePenWidth, PAT_STYLE thePenStyle);
```

Sets the pen properties of an object. theColor is the color of the line, thePenWidth is the width of the line in pixels, and thePenStyle is one of the three pen patterns shown in Table 5.

```
virtual void SetPenPattern(PAT_STYLE  
    thePenStyle);
```

Sets the pen style of an object, thePenStyle, to one of the patterns shown in Table 5.

```
virtual void SetPenWidth(short thePenWidth);
```

Sets the width of the pen's line, thePenWidth, in pixels.

```
virtual void SetPenColor(COLOR theColor, COLOR  
    theOffPenColor = COLOR_BLACK);
```

Sets the color of the pen's ink. theOffPenColor specifies the pen color when monochrome is being used.

```
PAT_STYLE GetPenPattern(void) const;
```

Returns the object's pen pattern. See Table 5.

```
short GetPenWidth(void) const;
```

Returns the object's pen width in pixels.

COLOR GetPenColor(void) const;

Returns the color of the pen's ink.

Brush Methods

The "brush" is the interior fill of closed shapes. The possible brush patterns are shown in Table 4. The open spaces of a pattern are drawn in the object's background color.

virtual void SetBrush(COLOR theColor, PAT_STYLE theBrushStyle);

Sets the ink color and the brush style of an interior fill.

virtual void SetBrushPattern(PAT_STYLE theBrushStyle);

Sets the brush pattern of an interior fill.

virtual void SetBrushColor(COLOR theColor, COLOR theOffBrushColor = COLOR_WHITE);

Sets the ink color of an interior fill. theOffBrushColor specifies the brush color when monochrome is being used.

PAT_STYLE GetBrushPattern(void) const;

Returns the brush pattern of the object.

COLOR GetBrushColor(void) const;

Returns the ink color of an interior fill.

Drawing Mode Methods

virtual void SetDrawingMode(DRAW_MODE theDrawMode);

Sets an object's drawing mode. See Table 1.

DRAW_MODE GetDrawingMode(void) const;

Returns the object's drawing mode.

Methods for Color/Monochrome

The environment class provides a facility for coding applications that will allow you to map colors correctly between monochrome and color monitors. The following methods change the mode (color or monochrome) and tell you what the mode is. Notice that these are static methods and thus affect all environment objects. A combination of monochrome and color environments within one application is not allowed.

```
BOOLEAN IsColorOn(void) const;
```

Returns a Boolean value of TRUE if DISPLAY is set to color and FALSE if it is not.

```
virtual void SetColorOn(BOOLEAN isColorOn);
```

Sets whether DISPLAY is color (TRUE) or monochrome (FALSE).

Other Utilities

```
void SetDrawingEnv(void) const;
```

Sets up the tools for drawing as they are defined in the object's environment.

```
void InvertColors(void);
```

Reverses the background and foreground colors defined in the current environment, including the colors of the brush and the pen.

Overrides

You should override none of the methods.

Summary: CEnvironment.h

```
#ifndef CEnvironment_H
#define CEnvironment_H

#include "PwrNames.h"
#include "CRect.h"
#include "CPoint.h"
#include "xvt.h"
#include "CGlobal.h"

extern FONT STDFONT;

class CWindow;

class CEnvironment
{
public:
    // Class utility:
    CEnvironment(COLOR theBackground = COLOR_WHITE,
        COLOR theForegroundColor = COLOR_BLACK,
        COLOR theBrushColor = COLOR_WHITE,
        PAT_STYLE theBrushPattern = PAT_SOLID,
        COLOR thePenColor = COLOR_BLACK,
        PAT_STYLE thePenPattern = PAT_SOLID,
        short thePenWidth = 1,
        const FONT& theFont = STDFONT,
        DRAW_MODE theDrawingMode = M_COPY);

    virtual ~CEnvironment(void);
```

```

    BOOLEAN IEnvironment(COLOR theBackground = COLOR_WHITE,
        COLOR theForegroundColor = COLOR_BLACK,
        COLOR theBrushColor = COLOR_WHITE,
        PAT_STYLE theBrushPattern = PAT_SOLID,
        COLOR thePenColor = COLOR_BLACK,
        PAT_STYLE thePenPattern = PAT_SOLID,
        short thePenWidth = 1,
        const FONT& theFont = STDFONT,
        DRAW_MODE theDrawingMode = M_COPY);

// Color utility:
virtual void SetColor(COLOR theBackground = COLOR_WHITE,
    COLOR theForegroundColor = COLOR_BLACK,
    COLOR theBrushColor = COLOR_WHITE,
    COLOR thePenColor = COLOR_BLACK,
    COLOR theOffBackground = COLOR_WHITE,
    COLOR theOffForegroundColor = COLOR_BLACK,
    COLOR theOffBrushColor = COLOR_WHITE,
    COLOR theOffPenColor = COLOR_BLACK);

COLOR GetForegroundColor(void) const;
virtual void SetForegroundColor(COLOR theColor, COLOR theOffColor =
    COLOR_WHITE);
COLOR GetBackgroundColor(void) const;
virtual void SetBackgroundColor(COLOR theColor, COLOR theOffColor =
    COLOR_BLACK);

// Fonts:
FONT* GetFont(void) const;
virtual void SetFont(const FONT& theFont,
    BOOLEAN isUpdate = FALSE);

// Pens:
virtual void SetPen(COLOR theColor, short thePenWidth,
    PAT_STYLE thePenStyle);
virtual void SetPenPattern(PAT_STYLE thePenStyle);
virtual void SetPenWidth(short thePenWidth);
virtual void SetPenColor(COLOR theColor, COLOR theOffPenColor =
    COLOR_BLACK);

PAT_STYLE GetPenPattern(void) const;
short GetPenWidth(void) const;
COLOR GetPenColor(void) const;

// Brushes:
virtual void SetBrush(COLOR theColor, PAT_STYLE theBrushStyle);
virtual void SetBrushPattern(PAT_STYLE theBrushStyle);
virtual void SetBrushColor(COLOR theColor, COLOR theOffBrushColor =
    COLOR_WHITE);
PAT_STYLE GetBrushPattern(void) const;
COLOR GetBrushColor(void) const;

// Drawing Mode:
virtual void SetDrawingMode(DRAW_MODE theDrawMode);
DRAW_MODE GetDrawingMode(void) const;

// Color-Monochrome
BOOLEAN IsColorOn(void) const;
virtual void SetColorOn(BOOLEAN isColorOn);

// Other Utilities:
void SetDrawingEnv(void) const;
void InvertColors(void);

```

```
private:
    FONT itsFont; // the system-wide font
    COLOR itsCBackgroundColor; // Color background
    COLOR itsCForegroundColor; // Color foreground
    COLOR itsBBackgroundColor; // Black&White background
    COLOR itsBForegroundColor; // Black&White foreground
    COLOR itsBrushColor; // Brush color
    COLOR itsPenColor; // Pen Color
    COLOR itsMonoPenColor; // Pen monochrome color
    COLOR itsMonoBrushColor; // Brush monochrome color
    PAT_STYLE itsBrushPattern; // the brush pattern
    short itsPenWidth; // the Pen Width
    PAT_STYLE itsPenPattern; // the Pen pattern
    DRAW_MODE itsDrawingMode; // the drawing mode

    static BOOLEAN itIsColor; // indicates whether DISPLAY is color
};

#endif CEnvironment_H
```

CError

CMem	CGlue	CSwitchBoard
CError	CDesktop	CStartup
	CEnvironment	CGlobalClassLib
	CGlobal	CGlobalUser
		CResourceMgr
		CPrintMgr

Description

CError is currently a very basic error reporting utility. In the future it will become a full-fledged error handling facility that uses some of the C++ 3.0 AT&T release features for error handling. Currently, CError defines a macro called PWRAssert, which reports XVT-Power++ errors.

Heritage

Superclass: none.

Usage

Use PWRAssert to verify that a certain condition is true. For example, you can call it as follows:

```

:::
aCircle = new CCircle();
PWRAssert(aCircle != NULL, 1234, "Failed to create
         "circle");
...

```

If the condition (aCircle!=NULL) is false, a fatal note pops up indicating the error number (1234) as well as the file and line number of the error.

The message ("Failed ...") is simply a comment by PWRAssert; it is not used and does not take up memory. The XVT-Power++ PWRERROR utility creates document files listing all error numbers and comments assigned to them for reference.

Compiling with PWRNoError option suppresses all assertions. Even the testing of the condition is suppressed, so be careful to ensure that

your program does not depend on any side effects that are a product of evaluating the condition.

Override

Override the contents of PWRError to handle errors as needed.

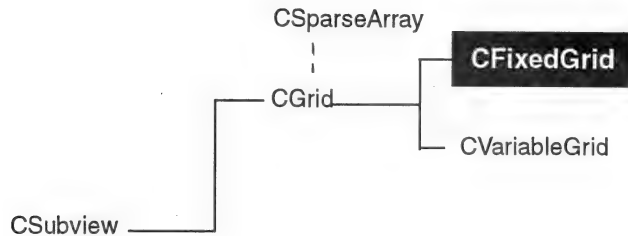
Summary: CError.h

```
#ifndef CError_H
#define CError_H

inline void PWRError(long theErrorId, char* theFile, int theLine)
{
    XVT_fatal("PWR Internal Error: %d, %s, %d", theErrorId, theFile,
              theLine);
}

#ifndef PWRNoError
#define PWRAssert(theCondition, theErrorId, theComment) if (!(theCondition))
PWRError(theErrorId, __FILE__, __LINE__)
#else
#define PWRAssert(cond, errorId, msg)
#endif PWRNoError
#endif CError_H
```

CFixedGrid



Description

CFixedGrid objects provide the means to arrange other CSubview objects into defined grid cells. All CFixedGrid cells are homogeneous in size. For more information, see CGrid and CVariableGrid.

Heritage

Superclass: CGrid

Subclasses: None

Usage

Create a fixed grid, initialize it, and insert CView objects into its cells. Objects inside a grid can either clip to their enclosing cell or force the cell to enlarge to incorporate the largest object.

All row and column operations assume that 0,0 is the coordinate for the top-left cell.

Environment

The lines of a fixed grid are drawn with the *pen*, and the area within each cell is painted with the *brush*. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width.

Data Members

None.

Public Methods

Constructor, Destructor, and Initializer Methods

```
CFixedGrid(CSubview* theEnclosure, const CRect&
theRegion, int theNumberOfRows, int
theNumberOfColumns);
```

A constructor. *theEnclosure* is a pointer to the subview that will contain the grid. *theRegion* is a coordinate location, local to the enclosure, that is used to place the grid. This method also requires you to specify the number of rows and columns in the grid, and it divides the given region into rows and columns.

```
CFixedGrid(CSubview* theEnclosure,
CPoint& theTopLeftCorner,
UNITS theRowHeight,
UNITS theColumnWidth,
int theNumberOfRows,
int theNumberOfColumns);
```

A constructor, which, like the preceding constructor, takes an enclosure for the grid and requires you to specify the number of rows and columns. Unlike the preceding constructor, it takes a *CPoint* coordinate for placing the top-left corner of the grid. You specify the size of the grid by giving it a row height and a column width, as well as the number of rows and columns.

```
CFixedGrid(const CFixedGrid& theGrid);
```

A copy constructor that duplicates the attributes, but not the contents, of a grid. That is, it copies the grid but does not do a deep copy of the objects it contains.

```
CFixedGrid& operator=(const CFixedGrid&
theGrid);
```

An assignment operator that duplicates the attributes, but not the contents, of a grid. That is, it copies the grid but does not do a deep copy of the objects it contains.

```
virtual ~CFixedGrid(void);
```

The destructor. It cleans up after the grid and deletes any subviews it contains.

```
BOOLEAN IFixedGrid(BOOLEAN isClipping      = TRUE,
PLACEMENT thePlacement = TOPLEFT,
BOOLEAN isGridVisible  = FALSE,
POLICY theSizingPolicy = ADJUSTCellSize,
BOOLEAN isVisible      = TRUE,
GLUETYPE theGlue       = NULLSTICKY);
```

The initializer. The `isClipping` parameter takes a Boolean value indicating whether the items placed in the grid are clipped to their cells. By default, clipping is turned off. `thePlacement` specifies the orientation for placing items in their cells—in the top left corner by default. `isGridVisible` sets whether the lines of the grid are visible or not. `theSizingPolicy` takes a value indicating one of the two different sizing policies for grids: `ADJUSTCellSize` (the default) or `ADJUSTCellNumber`. The `isVisible` parameter sets the visibility state for the entire grid object (as opposed to `isGridVisible`, which pertains to the horizontal and vertical lines within the grid). Finally, this initializer takes a glue type. You can override one or more of these defaults by calling `IFixedGrid`.

Cell Operations

`CFixedGrid` overrides the following basic inherited methods.

```
virtual CRect GetCellSize(int theRow, int  
theColumn) const;
```

Gets the size of the cell located at the given row and column. It returns a `CRect`, which consists of that cell's coordinates, which are local to the grid. All of the cells in a fixed grid are of the same size although their positions in the rows and columns of the grid differ.

```
virtual int GetRow(UNITS theHorizontalPoint)  
const;
```

Gets a row of the grid. It takes a horizontal location in coordinates relative to the grid and returns an integer indicating what row contains the given point.

```
virtual int GetCol(UNITS theVerticalPoint)  
const;
```

Gets a column of the grid. It takes a vertical location in coordinates relative to the grid. It returns an integer indicating which column contains the `VerticalLocation`.

Sizing Methods

`CFixedGrid` overrides the following basic inherited methods.

```
virtual void SizeCell(UNITS theNewCellWidth,  
UNITS theNewCellHeight);
```

Resizes each cell of the grid according to the given width and height, which has the effect of enlarging or reducing the size of the entire grid.

```
virtual void SetNumCells(int  
    theNewColumnNumber, int theNewRowNumber);
```

Enlarges or reduces the size of the entire grid by giving it the specified number of columns and number of rows.

```
virtual UNITS GetWidth(int col = 0) const;
```

Returns the width, in logical units, of the specified column; the leftmost column takes the number zero (0). The value of the column does not matter for fixed grids.

```
virtual UNITS GetHeight(int row = 0) const;
```

Returns the height, in logical units, of the specified row; the topmost row takes the number zero (0). The value of the row does not matter for fixed grids.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

```
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
```

```
virtual void Deactivate(void);
```

```
virtual void Disable(void);
```

```
virtual void DoCommand(long theCommand, void* theData=NULL);
```

```
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
```

```
virtual void DoTimer(long theTimerId);
```

```
virtual void DoUser(long theUserId, void* theData);
```

```
virtual void Draw(void);
```

```
virtual void Enable(void);
```

```
virtual CView* FindHitView(const CPoint& theLocation) const;
```

```
virtual CRect GetClippedFrame(void) const;
```

```
virtual long GetCommand(void) const;
```

```
virtual CWindow* GetCWindow(void) const;
```

```
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
```

```
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CSubview

```
virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
```

```

virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CGrid

```

virtual void AdjustCells(ADJUST theAdjustment);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoPlaceView(void);
virtual void DoSize(const CRect& theNewSize);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* GetContents(CPoint theLocation) const;
virtual CView* GetContents(int theRow, int theCol) const;
CView* GetHitItem(CPoint theLocation) const;
virtual int GetNumCols(void) const;
virtual int GetNumRows(void) const;
virtual PLACEMENT GetPlacement(void) const;
virtual void GetPosition(const CView* theView, int *theRow, int *theCol) const;
virtual void GetPosition(const CRect& theView, int *theRow, int *theCol) const;
virtual POLICY GetSizingPolicy(void) const;
virtual void HideGrid(void);
virtual void Insert(CView* theView);
virtual void Insert(CView* theView, int theRow, int theColumn);
virtual BOOLEAN IsClipping(void);
virtual BOOLEAN IsGridVisible(void);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void PlaceView(CView* theView, int theRow, int theCol);

```

```

virtual CView* Remove(int theRow, int theColumn);
virtual BOOLEAN Remove(const CView* theView);
void RemoveSubview(const CView* theView);
virtual CView* Replace(CView* theView);
virtual CView* Replace(CView* theView, int theRow, int theColumn);
void ResetCell(CView* theView, int theNewRow=-1, int theNewCol=-1);
virtual void SetClipping(BOOLEAN isClipping);
void SetDragPoint(const CPoint& theLocation);
virtual void SetPlacement(PLACEMENT thePlacement);
virtual void SetSizingPolicy(POLICY thePolicy);
virtual void ShowGrid(void);
virtual void Size(const CRect& theNewSize);
virtual void SizeCell(double theCellWidth, double theCellHeight) = NULL;
virtual BOOLEAN Validate(int theRow, int theCol);

```

Summary: CFixedGrid.h

```

#ifndef CFixedGrid_H
#define CFixedGrid_H

#include "CGrid.h"

class CFixedGrid : public CGrid
{
public:
    // Create, Initialize, Destruct:
    CFixedGrid(CSubview* theEnclosure, const CRect& theRegion,
               int theNumberOfRows, int theNumberOfColumns);

    CFixedGrid(CSubview* theEnclosure,
               CPoint& theTopLeftCorner,
               UNITS theRowHeight,
               UNITS theColumnWidth,
               int theNumberOfRows,
               int theNumberOfColumns);

    CFixedGrid(const CFixedGrid& theGrid);
    CFixedGrid& operator=(const CFixedGrid& theGrid);
    virtual ~CFixedGrid(void);

    BOOLEAN IFixedGrid(BOOLEAN isClipping = TRUE,
                       PLACEMENT thePlacement = TOPLEFT,
                       BOOLEAN isGridVisible = FALSE,
                       POLICY theSizingPolicy = ADJUSTCellSize,
                       BOOLEAN isVisible = TRUE,
                       GLUTYPE theGlue = NULLSTICKY);

    // Cell Operations:
    virtual CRect GetCellSize(int theRow, int theColumn) const;
    virtual int GetRow(UNITS theHorizontalPoint) const;
    virtual int GetCol(UNITS theVerticalPoint) const;

```



```
// Sizing:
virtual void SizeCell(UNITS theNewCellWidth, UNITS theNewCellHeight);
virtual void SetNumCells(int theNewColumnNumber, int theNewRowNumber);
virtual UNITS GetWidth(int col = 0) const;
virtual UNITS GetHeight(int row = 0) const;

protected:

private:

};

#endif CFixedGrid_H
```

CGlobal

CMem	CGlue	CSwitchBoard
CError	CDesktop	CStartup
	CEnvironment	CGlobalClassLib
	CGlobal	CGlobalUser
		CResourceMgr
		CPrintMgr

Description

CGlobal is a file that contains definitions used by XVT-Power++.

Heritage

Superclass: none

Usage

This file is already included where it is needed, and you should make no changes to it.

Glue Types

The following defines can be used to describe types of glue. See CGlue.

```
typedef long GLUETYPE;
#define LEFTSTICKY 0xFF000000L
#define TOPSTICKY 0x00FF0000L
#define RIGHTSTICKY 0x0000FF00L
#define BOTTOMSTICKY 0x000000FFL
#define SAMESTICKY 0xF0F0F0F0L
#define ALLSTICKY 0xFFFFFFFFL
#define TOPLEFTSTICKY 0xFFFF0000L
#define TOPRIGHTSTICKY 0x00FFFF00L
#define BOTTOMLEFTSTICKY 0xFF0000FFL
#define BOTTOMRIGHTSTICKY 0x0000FFFFL
#define NULLSTICKY 0x00000000L
```

Default Parameters

The following two defines are used to indicate certain default parameter values in some XVT-Power++ methods:

```
#define LAST 0x0000FFFF // Last item in a list
#define SAME 0xF0F0F0F0 // Use the previously-defined value
```

Internal XVT-Power++ Commands

XVT-Power++ generates the following commands internally. Note that the XVT-Power++ command base is 20,000 and therefore the numbers of all user-defined commands should be lower than this base.

```
#define PWRCommandBase 20000

#define NOWINDOWCmd (PWRCommandBase + 1) // all windows are closed
#define NATIVEViewCmd (PWRCommandBase + 2) //trapped by CScroller
#define NULLCmd (PWRCommandBase + 3) // default command setting

#define WFSelectCmd (PWRCommandBase + 4) /* wireframe selction */
#define WFDeselectCmd (PWRCommandBase + 5) /*wireframe
deselection */
#define WFSIZECmd (PWRCommandBase + 6) /* wireframe move/size */
```

Internal XVT-Power++ ID Numbers

XVT-Power++ assigns the following ID numbers internally. Note that the XVT-Power++ ID number base is 20,000. Therefore, all user-defined ID numbers should be lower than this base.

```
#define PWRIIDBase 20000
#define NULLid (PWRIIDBase + 1) //Default ID for objects with no ID
#define TASKDOCid (PWRIIDBase + 2) // ID number of the task document
```

Directional Flags

The following defines are used throughout XVT-Power++ to select directions or to designate the sides of an object. For example, scrollbars are oriented in horizontal or vertical directions, and objects such as radio buttons can be placed horizontally or vertically within their enclosure.

```
#define PWRTOP 0x000F
#define PWRBOTTOM 0x00F0
#define PWRLEFT 0x0F00
#define PWRRIGHT 0xF000

typedef enum {VERTICAL, HORIZONTAL} DIRECTION;
typedef DIRECTION SBTYPE;
```

Error Handling

```
#include "CError.h"
```

Internal XVT-Power++ Strings

XVT-Power++ defines NULLString to be equivalent to an empty string.

```
extern CString NULLString;
```

Min and Max (not defined by some compilers)

```
#ifndef min
#define max(x,y) ((x) > (y) ? (x) : (y))
#define min(x,y) ((x) < (y) ? (x) : (y))
#endif // min
```

Internal XVT-Power++ Resources

XVT-Power++ defines the following resources internally. Note that the XVT-Power++ resource ID base is 20,000. Therefore, all user-defined resource ID numbers should be lower than this base.

```
#define PWRRidBase 20000
#define NULLwidth 32
#define NULLheight 30
#define NULLicon (PWRRidBase + 1)
```

Units

```
typedef float UNITS;
```

Native Dialog Definitions

```
#define NATIVEDialogType // Native dialog window XVT type
#define NATIVEDialogAttr // Native dialog window XVT attributes
```

CGlobalClassLib

CMem	CGlue	CSwitchBoard
CError	CDesktop	CGlobalClassLib
	CEnvironment	CGlobalUser
	CGlobal	CResourceMgr
		CPrintMgr

Description

CGlobalClassLib is a privately defined class that contains all variables that are global to the class library. A similar class, CGlobalUser, exists for user global variables. Once it is instantiated, the CGlobalClassLib object acts as a receptacle for several kinds of items: handles for global objects, global attributes of the application, and global flags.

Heritage

Superclass: none

Usage

A CGlobalClassLib object is automatically instantiated by CApplication. Any class in the XVT-Power++ application framework hierarchy has access to the GlobalClassLib object through the G pointer provided by CBoss.

Do not add any application-specific information to CGlobalClassLib. Use the CGlobalUser class for those purposes.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

Global Objects

CDesktop	*itsDesktop;	the global desktop manager
CPrintMgr	*itsPrintMgr;	the global print manager
CApplication	*itsApplication	the application object

Global Flags

BOOLEAN	itIsTerminating;	whether the program is in termination state
BOOLEAN	itIsTextEvent;	whether to update text editing boxes

Global Object ID Count

long	itsIdCount;
------	-------------

Friends

friend class	CApplication;
friend class	CSwitchBoard;

Public Methods

Access to Global Objects

CDesktop* GetDesktop(void);

Returns a pointer to the global desktop object.

void SetDesktop(CDesktop* theNewDesktop);

Takes a pointer to a new desktop, theNewDesktop. Once you pass in the desktop, the CGlobalClassLib object assumes ownership of that object. This means that when the CGlobalClassLib object is deleted, it deletes the new desktop object as well.

CPringMgr* GetPrintMgr(void);

Returns a pointer to a CPrintMgr object. When you get this pointer, you should just use it and not delete it.

void SetPrintMgr(CPrintMgr* theNewPrintManager);

Takes a pointer to a new CPrintMgr object. Each application has a global print manager that takes care of the application's

printing needs. This global object can be accessed through the CGlobalClassLib object. If you were to implement your own print manager, you would set the global print manager by calling SetPrintMgr. As is true for CGlobalClassLib::SetDesktop, if you actually set the print manager by passing a pointer to it, CGlobalClassLib assumes responsibility for this object and deletes it when the application terminates. The print manager should be deleted nowhere else.

CApplication* GetApplication(void);

Returns a pointer to the application object.

CTaskWin* GetTaskWin(void);

Returns a pointer to the application's task window. For details on the task window, see CTaskWin.

CWindow* GetHiddenWindow(void);

Returns a pointer to the XVT Portability Toolkit "hidden window," which is a window that is created when you call GetHiddenWindow. This window displays the XVT Portability Toolkit information to which you may want access. For example, if you want information on font metrics before any windows are created, you would call this method and read about font metrics.

Access to Global Flags

BOOLEAN IsTerminating(void);

Returns TRUE or FALSE depending on whether the application is undergoing a shutdown. This method is provided for some classes that must modify their behavior, or their response to events, when the application is terminating.

BOOLEAN IsUpdatingTextEdits(void);

Returns TRUE or FALSE depending on how the itIsTextEvent flag is set. A value of TRUE means that every time an event is received—an update event or a character event and so on—it is passed down to the text edit system. Sometimes updating is turned off, as when text boxes are being moved or dragged or when the user validates and wants to override the key that has been pressed and intended for a certain text edit box.

void UpdateTextEdits(BOOLEAN isUpdating);

Sets the value of the itIsTextEvent flag to a Boolean value of TRUE or FALSE, which determines whether text boxes are updated.

Global ID Count

XVT-Power++ID GetId(void);

XVT-Power++ keeps a global count, inside CGlobalClassLib, that is used to supply fresh, unique ID numbers to objects that require IDs. This method returns a unique ID each time it is called. During the execution session of an application, GetId will never return the same number twice.

Private Methods

Constructor and Destructor Methods

CGlobalClassLib(CApplication*theApplication);

The constructor, which takes a pointer to an application.

virtual ~CGlobalClassLib(void);

The destructor, which cleans up after the desktop. Whenever you set a new desktop, you must ensure that this desktop is not deleted anywhere else.

Summary: CGlobalClassLib.h

```
#ifndef CGlobalClassLib_H
#define CGlobalClassLib_H

#include "PwrNames.h"
#include "xvt.h"
#include "CGlobal.h"

class CApplication;
class CEnvironment;
class CDesktop;
class CTaskWin;
class CPrintMgr;
class CWindow;

typedef int PWRID;

class CGlobalClassLib
{
public:
    // Access to global objects:
    CDesktop* GetDesktop(void);
    void SetDesktop(CDesktop* theNewDesktop);

    CPrintMgr* GetPrintMgr(void);
    void SetPrintMgr(CPrintMgr* theNewPrintManager);

    CApplication* GetApplication(void);

    CTaskWin* GetTaskWin(void);

    CWindow* GetHiddenWindow(void);
};
```



```
// Access to global flags:
BOOLEAN IsTerminating(void);
BOOLEAN IsUpdatingTextEdits(void);
void UpdateTextEdits(BOOLEAN isUpdating);

//Global Id Count:
PWID GetId(void);

private:
    friend class CApplication;
    friend class CSwitchBoard;

    // constructor and destructor
    CGlobalClassLib(CApplication *theApplication);
    virtual ~CGlobalClassLib(void);

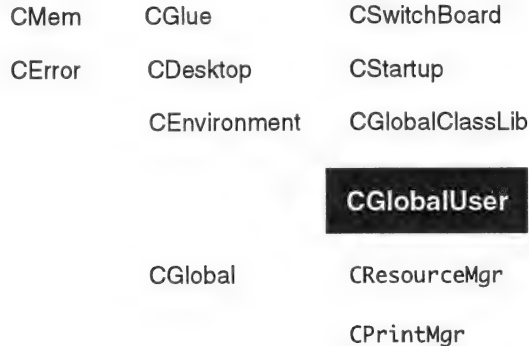
    // Global objects:
    CDesktop *itsDesktop; // the global desktop manager
    CPrintMgr* itsPrintMgr; // the global print manager
    CApplication* itsApplication; // the application object

    // Global flags:
    BOOLEAN itIsTerminating; // is the application terminating
    BOOLEAN itIsTextEvent; // update text editing boxes?

    // Global object Id Count:
    long itsIdCount;
};

#endif CGlobalClassLib_H
```

CGlobalUser



Description

CGlobalUser is a base class by which XVT-Power++ users provide application framework objects with a reference to application-specific global information/objects. It is a model of CGlobalClassLib for users. You can insert your own references to global objects or flags or attributes.

Heritage

Superclass: none

Usage

The use of this class is optional; it is available as a hook. This class instantiated is by the user's application object and given to CApplication::IApplication to process. The variables are thereafter accessed through CBoss's GU pointer.

Public Methods

```
CGlobalUser(void);
```

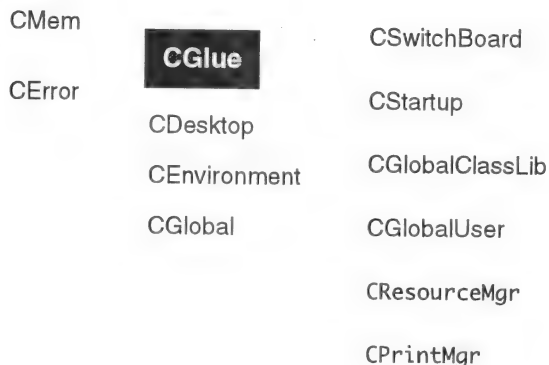
The constructor.

Summary: CGlobalUser.h

```
#ifndef CGlobalUser_H
#define CGlobalUser_H
#include "PwrNames.h"
```

```
class CGlobalUser
{
    public:
        CGlobalUser(void) {}
};
#endif CGlobalUser_H
```

CGLue



Description

CGLue allows objects to have stickiness properties. When a view is resized a sticky object will, depending on its stickiness, stretch with the view or stay fixed by a constant distance from the view's borders.

Heritage

Superclass: none

Usage

Although CGLue is not a private class, it is used internally. Glue objects are created and destroyed internally. CGLue has been made public so that you can derive from it and set your own derived glue objects for any view.

The gluing options, passed in the GLUETYPE variable, are chosen by ORing one or more of the following:

TOPSTICKY	BOTTOMRIGHTSTICKY
BOTTOMSTICKY	TOPLEFTSTICKY
LEFTSTICKY	TOPRIGHTSTICKY
RIGHTSTICKY	BOTTOMLEFTSTICKY
ALLSTICKY	

For example, an object whose glue is set as follows remains glued to the top-right corner of its enclosing border as the window is stretched:

SetGlue(TOPSTICKY|RIGHTSTICKY)

However, an object whose stickiness is defined as follows stretches or shrinks as the window changes size:

SetGlue(ALLSTYCKY)

Public Data Members

None.

Protected Data Members

None.

Private Data Members

GLUETYPE	itsGlue;	its conjoined sticky properties
UNITS	itsTopScaling;	distance to itsBorder's top
UNITS	itsLeftScaling;	distance to itsBorder's left
UNITS	itsBottomScaling;	distance to itsBorder's bottom
UNITS	itsRightScaling;	distance to itsBorder's right
CView*	itsOwner;	the view owning this glue

Public Methods

CGLue(CView* theOwner);

A constructor. It takes a pointer to the view associated with this glue object (theOwner).

CGLue(const CGLue& theGlue);

A copy constructor. It disallows copy construction.

CGLue& operator=(const CGLue& theGlue);

An assignment operator that makes two glue objects identical in their glue properties. It does not change the owners of the glue objects.

virtual ~CGLue(void);

The destructor, which cleans up after the glue object.

virtual void SetGlue(GLUETYPE theGlue);

Sets the glue type of a glue object. By default, a glue object has a type of NULLSTICKY upon creation. See the table under "Usage" for a listing of valid glue types.

```
virtual GLUETYPE GetGlue(void) const;
```

Returns the glue type of a glue object.

Glue Event Handling

```
virtual void SizeOwner(void);
```

Sizes a view according to its glue type when the view's enclosure is sized. SizeOwner is called internally. This method is called by XVT-Power++ whenever necessary, and you can override it to implement different gluing behaviors.

Overrides

GetFrame and SetFrame.

Summary: CGLue.h

```
#ifndef CGLue_H
#define CGLue_H

#include "PwrNames.h"
#include "xvt.h"
#include "CGlobal.h"

class CRect;
class CView;

class CGLue
{
public:
    CGLue(CView* theOwner);
    CGLue(const CGLue& theGlue);
    CGLue& operator=(const CGLue& theGlue);
    virtual ~CGLue(void);

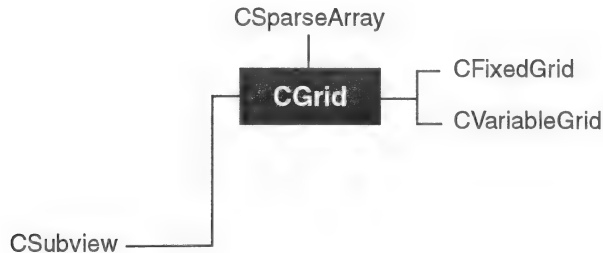
    virtual void SetGlue(GLUETYPE theGlue);
    virtual GLUETYPE GetGlue(void) const;

    // Glue event handling:
    virtual void SizeOwner(void);

private:
    GLUETYPE itsGlue; // its conjoined sticky properties
    UNITS itsTopScaling; // distance to itsBorder's top
    UNITS itsLeftScaling; // distance to itsBorder's left
    UNITS itsBottomScaling; // distance to itsBorder's bottom
    UNITS itsRightScaling; // distance to itsBorder's right
    CView* itsOwner; // the View owning this glue
};

#endif CGLue_H
```

CGrid



Description

CGrid is an abstract class that provides the means to arrange other CView objects into defined grid cells.

A CGrid in its abstract sense is simply an area of the screen that has been divided into rows and columns. These are the only assumptions made at the CGrid level. Each view object contained in the grid is placed inside an individual grid cell. A *cell* is a rectangular location comprised of the intersection of a certain row with a certain column. Objects inserted into cells can either be clipped to their cells or simply placed inside. If the objects are not clipped and are too big to fit into their cells they overlap from cell to cell. An object can be placed inside a cell relative to its top left, top right, bottom left, or bottom right corner.

When rows and columns are being counted in a grid, the topmost row and the leftmost column are both counted as zero (0); thus, counting proceeds from zero.

Depending on its attributes, a grid can have the following characteristics:

- the lines in the grid and/or the entire grid can be either visible or invisible.
- objects in the grid can be snapped into a cell as the user moves/resized them.
- objects can be clipped to their cells.
- cells can be resized to enclose the largest object.
- an item inserted into a grid snaps to a certain corner of its cell and can be centered within the cell.

- the grid can be sized either by adding and removing cells or by sizing the cells.

There are two different sizing policies for grids. One policy sizes the grid by sizing its cells; the number of cells remains constant and the size of the cells is adjusted as necessary when the grid is enlarged or shrunk. A second sizing policy adjusts the number of cells, keeping the size of the cells constant while adding cells to enlarge the grid or removing cells to shrink it.

Heritage

Superclass: CSubview

Subclasses: CFixedGrid, CVariableGrid

Usage

This is a virtual base class that is used to implement two different types of grids: grids with cells of homogeneous size (CFixedGrid) and grids with cells of heterogeneous size (CVariableGrid). Thus, objects of this class cannot be instantiated directly. All row and column operations assume that 0,0 is the coordinate of the top-left cell.

For any view created using a grid as its enclosure, you must call one of the insertion methods (Insert, Replace) to indicate the cell in which it belongs.

Enums

These enums are used by the CGrid methods:

ADJUST

MAXIMIZE	adjust cell size to equal the size of the largest view
MINIMIZE	adjust cell size to equal the size of the smallest view

PLACEMENT

TOPLEFT	give contents top-left justification
TOPRIGHT	give contents top-right justification
BOTTOMLEFT	give contents bottom-left justification
BOTTOMRIGHT	give contents bottom-right justification

Policy

ADJUSTCellSize	handle sizing events by resizing each individual cell
ADJUSTCellNumber	handle sizing events by adding or removing cells

Public Data Members

None.

Protected Data Members

UNITS	itsCellWidth;	the width of each cell
UNITS	itsCellHeight;	the height of each cell
int	itsRowNumber;	the number of rows
int	itsColumnNumber;	the number of columns
POLICY	itsSizingPolicy;	type of size behavior
CSparseArray*	itsCells;	matrix of cells

Private Data Members

BOOLEAN	itIsClipping;	whether an item clips to a cell
PLACEMENT	itsPlacement;	snapping corner
CPoint	itsPlacementOffset;	snap corner offset
CPoint*	itsDragPoint;	snapping point for mouse drags
int	itsDragState;	keeps track of state
BOOLEAN	itIsHit;	keeps track of mouse event state
BOOLEAN	itIsGridVisible;	whether the grid lines are drawn

Public Methods**Destructor and Initializer Methods**

Because CGrid is an abstract class, it has no public constructors.

```
virtual ~CGrid(void);
```

The destructor. It cleans up after the grid and deletes any views nested within it.

```

BOOLEAN IGrid(BOOLEAN isClipping           = FALSE,
               PLACEMENT thePlacement       = TOPLEFT,
               BOOLEAN isGridVisible        = FALSE,
```

```

POLICY theSizingPolicy      = ADJUSTCellSize,
BOOLEAN isVisible          = TRUE,
GLUETYPE theGlue           = NULLSTICKY);

```

The initializer. The `isClipping` parameter takes a Boolean value indicating whether the items placed in the grid are clipped to their cells. By default, clipping is turned off. `thePlacement` specifies the orientation for placing items in their cells—in the top left corner by default. `isGridVisible` sets whether the lines of the grid are visible or not. `theSizingPolicy` takes a value indicating one of the two different sizing policies for grids: `ADJUSTCellSize` (the default) or `ADJUSTCellNumber`. The `isVisible` parameter sets the visibility state for the entire grid object (as opposed to `isGridVisible`, which pertains to the horizontal and vertical lines within the grid). Finally, this initializer takes a glue type.

Inherited Drawing Methods

```

virtual void DoDraw(const CRect&
    theClippingRegion);

```

A method that is inherited from the `CSubview` level. Its behavior is the same as that of `CSubview`'s `DoDraw`. It takes care of any drawing the grid must do and draws all of the its subviews, ensuring that all of the subviews are clipped to the grid. `theClippingRegion` is the part of the grid that needs to be refreshed and is in global, window-relative coordinates.

```

virtual void Draw(const CRect&
    theClippingRegion);

```

Takes care of any drawing the grid must do. `theClippingRegion` is the part of the grid that needs to be refreshed.

Clipping Methods

```

virtual void SetClipping(BOOLEAN isClipping);

```

Sets the state of the `ItIsClipping` data member to `TRUE` or `FALSE`, which determines whether items in the grid clip to their cells.

```

virtual BOOLEAN IsClipping(void);

```

Returns `TRUE` or `FALSE`, indicating whether the objects contained in the grid are clipped to their cells.

Placement Policy Methods

```

virtual void SetPlacement(PLACEMENT
    thePlacement);

```

Sets the placement policy for the grid; that is, which of the four cell corners is to be used as the orientation for placing an item inside a cell.

```
virtual PLACEMENT GetPlacement(void) const;
```

Returns the current placement policy of the grid—top left, top right, bottom left, or bottom right.

Visibility Methods

```
virtual void ShowGrid(void);
```

Reveals the row and column lines of the grid.

```
virtual void HideGrid(void);
```

Makes the row and column lines of the grid invisible.

```
virtual BOOLEAN IsGridVisible(void);
```

Returns a Boolean value indicating whether the row and column lines of the grid are visible.

Removal Functions

```
virtual CView* Remove(int theRow, int  
theColumn);
```

Removes an item from the grid when you give it a row number and a column number for the item. It returns a pointer to the view that was removed; if no view is found in the given row and column coordinate, this method returns NULL.

```
virtual BOOLEAN Remove(const CView* theView);
```

Removes the specified view, theView, from the grid and returns a Boolean value of TRUE if it is successful.

Placement Methods

The placement methods, which must be called for each subview, include both *insert* and *replace* functions. The replace methods are similar to the insert methods, with an importance difference. The insert function inserts the object into the cell even if the cell is occupied. Thus, there can be more than one view per cell. The replace methods remove any object already occupying the cell and replace it with the new object, returning a pointer to the object that was removed. If there is no object to be removed, the replace methods return NULL.

```
virtual void Insert(CView* theView, int theRow,  
int theColumn);
```

Inserts the designated view into the given row and column of the grid. It takes a pointer to the view that is to be inserted, a row number, and a column number. The coordinates of the view are adjusted so that it can be placed into the cell.

```
virtual void Insert(CView* theView);
```

Takes a pointer to a view, examines the view's coordinates, and calculates the row and column of the grid into which it fits best.

```
virtual CView* Replace(CView* theView, int  
theRow, int theColumn);
```

Takes a pointer to a view and places it inside the cell that is located at the given row and column, removing any object already occupying the cell and returning a pointer to the removed object.

```
virtual CView* Replace(CView* theView);
```

Takes a pointer to a view, examines the view's coordinates, and calculates the row and column of the grid into which it fits best. It removes any object that is already occupying the cell and returns a pointer to the removed object.

Cell Get Operations

```
virtual CView* GetContents(int theRow, int  
theCol) const;
```

Gets the contents of the cell (a view) located at the given row and column and returns a pointer to the view. If the cell has no contents, this method returns NULL.

```
virtual CView* GetContents(CPoint theLocation)  
const;
```

Gets the contents of the cell (a view) at theLocation, a coordinate that is local to the grid. This method returns a pointer to the view. If the cell has no contents, this method returns NULL.

```
virtual int GetNumRows(void) const;
```

Returns the number of rows in the grid.

```
virtual int GetNumCols(void) const;
```

Returns the number of columns in the grid.

```
virtual void GetPosition(const CView* theView,  
int *theRow, int *theCol) const;
```

Gets the position of a certain view. It takes a pointer to that view (location) and it takes two pointers to some integers that are filled in by GetPosition, the row and the column. After this

method is called, the row and the column designated by the integers will contain the location of that view, if the view happens to be in the grid. If the view is not contained in the grid, then the row and the column are set to -1.

```
virtual void GetPosition(const CRect& theView,  
int *theRow, int *theCol) const;
```

Gets the grid position of a certain region. You pass it a rectangular area (set of coordinates) relative to the grid and ask it which row and column this rectangular area belongs to. It returns the row and column or -1 if the rectangular area does not match any of the row and column coordinates.

```
virtual CRect GetCellSize(int theRow, int  
theCol) const = NULL;
```

Gets the size of the cell located at the given row and column. It returns a CRect, which consists of that cell's coordinates, which are local to the grid.

```
virtual int GetRow(UNITS theHorizontalLocation)  
const = NULL;
```

Gets a row of the grid. It takes a horizontal location in coordinates relative to the grid and returns an integer indicating what row contains theHorizontalLocation.

```
virtual int GetCol(UNITS theVerticalLocation)  
const = NULL;
```

Gets a column of the grid. It takes a vertical location in coordinates relative to the grid. It returns an integer indicating which column contains theVerticalLocation.

```
virtual BOOLEAN Validate(int theRow, int  
theCol);
```

Takes a row and column and returns TRUE or FALSE, depending on whether the given row and column contains a valid grid cell. Validate checks the row and column to verify that they are valid in the grid. This method is called internally whenever a new item is being inserted into the grid. This method can be overridden. One possibility is to override Validate so that it creates a valid row and column to accommodate the inserted object if it does not find them and returns TRUE.

Inherited Sizing Methods

```
virtual void Size(const CRect& theNewSize);
```

Sizes the grid according to the coordinates of theNewSize. The sizing of the grid varies according to the sizing policy that has

been set. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, `theNewSize`, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated.

```
virtual void DoSize(const CRect& theNewSize);
```

Sizes the grid according to the coordinates of `theNewSize`. This method also sizes the grid's subviews according to their glue setting. The sizing of the grid varies according to the sizing policy that has been set.

Sizing (Varies Among Grid Types)

```
virtual void SizeCell(UNITS theCellWidth, UNITS  
theCellHeight) = NULL;
```

Resizes each cell of the grid according to the given width and height, which has the effect of enlarging or reducing the size of the entire grid.

```
virtual void SetNumCells(int  
theNewColumnNumber, int theNewRowNumber)=  
NULL;
```

Enlarges or reduces the size of the entire grid by giving it the specified number of columns and number of rows.

```
virtual void AdjustCells(ADJUST theAdjustment);
```

Takes the typedef `ADJUST`, which can have a value of either `MAXIMIZE` or `MINIMIZE`. `MAXIMIZE` goes through all of the objects contained in the grid, finds the largest one, and makes all of the grid cells that size. `MINIMIZE` goes through all of the objects contained in the grid, finds the smallest one, and makes all of the grid cells that size.

```
virtual UNITS GetWidth(int theColumn = 0) const  
= NULL;
```

Returns the width, in logical units, of the specified column; the leftmost column takes the number zero (0).

```
virtual UNITS GetHeight(int theRow = 0) const =  
NULL;
```

Returns the height, in logical units, of the specified row; the topmost row takes the number zero (0).

```
virtual POLICY GetSizingPolicy(void) const;
```

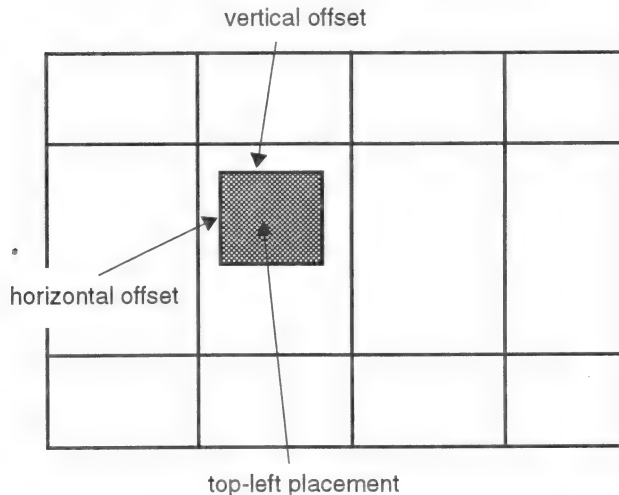
Gets the sizing policy for the grid; that is, whether cell size is adjusted as the grid is made larger/smaller or whether cell size remains constant while cells are added or removed.

virtual void SetSizingPolicy(POLICY thePolicy);

Sets the sizing policy for the grid; that is, whether cell size will be adjusted as the grid is made larger/smaller or whether cell size will remain constant while cells are added or removed. This method takes a value of either ADJUSTCellSize or ADJUSTCellNumber.

virtual void SetPlacementOffset(const CPoint& theNewPlacementOffset);

Sets the placement offset, which is used in conjunction with the placement policy to place items, as shown here:



virtual CPoint GetPlacementOffset(void);

Returns the placement offset.

Mouse Event Methods (All Inherited)

All of the following mouse methods override the corresponding methods on CSubview because grid snapping has been implemented for movable and sizable objects contained in a grid. *Snapping* means that as the user drags an object across a grid, it snaps to each cell; the object cannot be dragged halfway between one cell and another.

Mouse events are passed directly to the view at the location where the event occurs. Each of the following methods has a `theButton`

argument that specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right). By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button.

```
virtual voidMouseDown(CPoint theLocation,  
                      short theButton    = 0,  
                      BOOLEAN isShiftKey = FALSE,  
                      BOOLEAN isControlKey= FALSE);
```

When the user presses down a mouse button over a grid containing local coordinate `theLocation`, the grid receives and handles this event.

```
virtual voidMouseMove(CPoint theLocation,  
                     short theButton    = 0,  
                     BOOLEAN isShiftKey = FALSE,  
                     BOOLEAN isControlKey= FALSE);
```

When the user moves the mouse over a grid containing local coordinate `theLocation`, the grid receives and handles this event, whether the mouse button is down or not.

```
virtual voidMouseUp(CPoint theLocation,  
                   short theButton    = 0,  
                   BOOLEAN isShiftKey = FALSE,  
                   BOOLEAN isControlKey= FALSE);
```

When the user releases a mouse button over a grid containing local coordinate `theLocation`, the grid receives and handles this event. This event is not necessarily preceded by a `MouseDown` or `MouseMove` event because those button actions may have taken place outside the grid.

```
virtual voidMouseDouble(CPoint theLocation,  
                       short theButton    = 0,  
                       BOOLEAN isShiftKey = FALSE,  
                       BOOLEAN isControlKey=FALSE);
```

When the user double clicks a mouse button over a grid containing local coordinate `theLocation`, the grid receives and handles this event.

```
virtual CView* FindEventTarget(const CPoint&  
                              theLocation) const;
```

A method that is called to find the target of a mouse event within a subview. This virtual method is defined to return the deepest subview at the global coordinates of `theLocation`. `CGrid` overrides this method in order to trap all events regardless of the subviews it contains or to change the event targeting algorithm. For example, a list box contains many

subviews (text items). When you click on a text item inside a grid, the event does not go to the view. Instead, the grid traps the event by returning this (itself) as the target for the mouse events received. Then, it treats the mouse events as necessary, without sending them on to its subviews.

```
virtual CView* FindHitView(const CPoint&
theLocation) const;
```

Overrides the CView method of the same name, which returns the actual view that should receive mouse events. It returns the grid's wire frame if the grid is being sized or dragged.

```
void RemoveSubview(const CView*theView);
```

Removes a view, theView, from the grid's list of views.

Protected Methods

Constructors

The copy constructors and equal operators of all classes in the CView hierarchy, including the grid classes, copy all of a view's attributes—size, color, active state, whether it is movable or sizable—but they do not copy all the objects that are inside of it.

```
CGrid(CSubview* theEnclosure, const CRect&
theRegion);
```

A constructor. theEnclosure is a pointer to the subview that will contain the grid. theRegion is a coordinate location, local to the enclosure, that is used to place the grid.

```
CGrid(const CGrid& theGrid);
```

A copy constructor that duplicates the attributes, but not the contents, of a grid. That is, it copies the grid but does not do a deep copy of the objects it contains.

```
CGrid& operator=(const CGrid& theGrid);
```

An assignment operator that duplicates the attributes, but not the contents, of a grid. That is, it copies the grid but does not do a deep copy of the objects it contains.

Placement Methods

```
virtual void PlaceView(CView* theView, int
theRow, int theCol);
```

Takes a view and the row and column to which the view belongs and ensures that this view has been placed correctly within the

cell. This method is called internally when a view is inserted or replaced.

```
virtual void DoPlaceView(void);
```

Iterates through the objects contained in the grid, calling PlaceView for each one. This method is called internally when a grid is sized or moved.

Private Methods

```
void SetDragPoint(const CPoint& theLocation);
```

When an object is being dragged within a grid, sets the point where the object was first grabbed in order to implement grid snapping.

```
void ResetCell(CView* theView, int theNewRow=-1, int theNewCol=-1);
```

When an object is moved to a new cell within a grid, resets the cell to ensure it has the coordinates of its new location.

```
CView* GetHitItem(CPoint theLocation) const;
```

An internal method that is used for snapping. Given a certain location that is local to the grid's coordinates, it returns the item at that grid location. This method is similar to methods for finding subviews at the CSubview level.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

```
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
```

```
virtual void Deactivate(void);
```

```
virtual void Disable(void);
```

```
virtual void DoCommand(long theCommand, void* theData=NULL);
```

```
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
```

```
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Enable(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
```

```

virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);

```

```

virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

Summary: CGrid.h

```

#ifndef CGrid_H
#define CGrid_H

#include "CSubview.h"

class CSparseArray;

typedef enum
{
    MAXIMIZE,      // adjust cell size to equal the largest view
    MINIMIZE       // adjust cell size to equal the smallest view
}
ADJUST;
typedef enum
{
    TOPLEFT = PWRTOP|PWRLEFT,      //give contents top-left justification
    TOPRIGHT = PWRTOP|PWRRIGHT,    //give contents top-right justification
    BOTTOMLEFT = PWRBOTTOM|PWRLEFT, //give contents bottom-left justification
    BOTTOMRIGHT = PWRBOTTOM|PWRRIGHT // give contents bottom-right
                                   // justification
}
PLACEMENT;
typedef enum
{
    ADJUSTCellSize, // handle sizing events by resizing each individual cell
    ADJUSTCellNumber // handle sizing events by adding or removing cells
}
POLICY;

class CGrid : public CSubview
{
public:
    // Create, Initialize, Destruct:
    virtual ~CGrid(void);

```

```

    BOOLEAN IGrid(BOOLEAN isClipping = FALSE,
                  PLACEMENT thePlacement = TOPLEFT,
                  BOOLEAN isGridVisible = FALSE,
                  POLICY theSizingPolicy = ADJUSTCellSize,
                  BOOLEAN isVisible = TRUE,
                  GLUETYPE theGlue = NULLSTICKY);

// Graphic Utility Functions:
virtual void DoDraw(const CRect& theClippingRegion);
virtual void Draw(const CRect& theClippingRegion);

virtual void SetClipping(BOOLEAN isClipping);
virtual BOOLEAN IsClipping(void);

virtual void SetPlacement(PLACEMENT thePlacement);
virtual PLACEMENT GetPlacement(void) const;

virtual void ShowGrid(void);
virtual void HideGrid(void);
virtual BOOLEAN IsGridVisible(void);

// Removal Functions:
virtual CView* Remove(int theRow, int theColumn);
virtual BOOLEAN Remove(const CView* theView);

// Placement Functions: must be called for each subview
virtual void Insert(CView* theView, int theRow, int theColumn);
virtual void Insert(CView* theView);

virtual CView* Replace(CView* theView, int theRow, int theColumn);
virtual CView* Replace(CView* theView);

// Cell Get Operations:
virtual CView* GetContents(int theRow, int theCol) const;
virtual CView* GetContents(CPoint theLocation) const;
virtual int GetNumRows(void) const;
virtual int GetNumCols(void) const;

virtual void GetPosition(const CView* theView,
                        int *theRow, int *theCol) const;
virtual void GetPosition(const CRect& theView,
                        int *theRow, int *theCol) const;

virtual CRect GetCellSize(int theRow, int theCol) const = NULL;
virtual int GetRow(UNITS theHorizontalLocation) const = NULL;
virtual int GetCol(UNITS theVerticalLocation) const = NULL;

virtual BOOLEAN Validate(int theRow, int theCol);

// Sizing: (will vary between grid types)
virtual void Size(const CRect& theNewSize);
virtual void DoSize(const CRect& theNewSize);

virtual void SizeCell(UNITS theCellWidth, UNITS theCellHeight)=NULL;
virtual void SetNumCells(int theNewColumnNumber, int theNewRowNumber)=NULL;
virtual void AdjustCells(ADJUST theAdjustment);

virtual UNITS GetWidth(int theColumn = 0) const = NULL;
virtual UNITS GetHeight(int theRow = 0) const = NULL;

virtual POLICY GetSizingPolicy(void) const;
virtual void SetSizingPolicy(POLICY thePolicy);

virtual void SetPlacementOffset(const CPoint& theNewPlacementOffset);
virtual CPoint GetPlacementOffset(void);

```

```

// Movement of subviews:
virtual void MouseDown(CPoint theLocation,short theButton = 0,
                      BOOLEAN isShiftKey=FALSE,
                      BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,
                      BOOLEAN isShiftKey=FALSE,
                      BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0,
                    BOOLEAN isShiftKey=FALSE,
                    BOOLEAN isControlKey=FALSE);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,
                        BOOLEAN isShiftKey=FALSE,
                        BOOLEAN isControlKey=FALSE);

virtual CView* FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
void RemoveSubview(const CView*theView);

protected:
    UNITS itsCellWidth;           // the width of each cell
    UNITS itsCellHeight;         // the height of each cell
    int itsRowNumber;            // The number of rows
    int itsColumnNumber;         // The number of columns
    POLICY itsSizingPolicy;       // type of Size behavior
    CSparseArray* itsCells;       // matrix of cells

    CGrid(CSubview* theEnclosure, const CRect& theRegion);
    CGrid(const CGrid& theGrid);
    CGrid& operator=(const CGrid& theGrid);

    // Other:
    virtual void DoPlaceView(void);
    virtual void PlaceView(CView* theView, int theRow, int theCol);

private:
    BOOLEAN itIsClipping;         // Clip to cell?
    PLACEMENT itsPlacement;       // snapping corner
    CPoint itsPlacementOffset;    // snap corner offset
    CPoint* itsDragPoint;         // dragging snapping point
    int itsDragState;             // keep track of state
    BOOLEAN itIsHit;              // keep track of mouse event state
    BOOLEAN itIsGridVisible;      // draw the grid lines

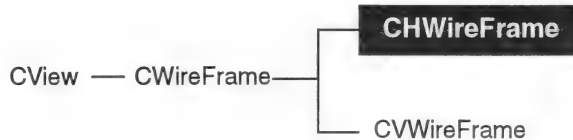
    void SetDragPoint(const CPoint& theLocation);
    void ResetCell(CView* theView, int theNewRow=-1,
                  int theNewCol=-1);

    CView* GetHitItem(CPoint theLocation) const;
};

#endif CGrid_H

```

CHWireFrame



Description

CHWireFrame is a class that overrides CWireFrame in order to implement special wire frame behavior: horizontal-only dragging. This behavior is achieved by trapping all MouseMove events and fixing the vertical position to a constant value.

Heritage

Superclass: CWireFrame

Usage

Give a view a horizontal wire frame as follows:

```
aView->SetWireFrame(new CHorizontalWireFrame(aView));
aView->SetDragging(TRUE);
```

Data Members

None.

Public Methods

```
CHorizontalWireFrame(CView*theOwner,CSubview*
theGroupEnclosure = NULL);
```

Calls the inherited constructor, that is, the constructor of CWireFrame. theOwner is a pointer to its owner, the CView object with which it is associated. It gets information about this object, such as the object's enclosure, from the object itself. theGroupEnclosure is the subview that acts as the enclosure of the group containing this wire frame.

```
virtual voidMouseMove(CPoint theLocation,short
theButton = 0, BOOLEAN isShiftKey=FALSE,
BOOLEAN isControlKey=FALSE);
```

An overridden method that suppresses vertical movement. When the user moves the mouse over a view containing local coordinate theLocation, the view receives and handles this

event. If the wire frame is in a selected state, it responds to the mouse move event by resizing or dragging the rubberband frame in a horizontal direction. theButton specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right). By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoActivate(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
```

```
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
```

```
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CWireFrame

```
void ClearGroupRegion(void);
virtual void Deselect(void);
virtual void DoAutoScroll(int theHorizontalIncrement, int theVerticalIncrement);
virtual BOOLEAN DoBorderCheck(const CPoint& theLocation);
virtual void DoDeselect(CView* theView);
virtual BOOLEAN DoDraggingCheck(const CPoint& theLocation);
virtual void Drag(CPoint theNewLocation);
virtual void Draw(const CRect& theClippingRegion);
virtual void DrawFrameGrabbers(DRAWSTATE isDrawing);
virtual void DrawWireFrame(DRAWSTATE isDrawing);
virtual CSubview* GetGroupEnclosure(void) const;
virtual CView* GetOwner(void) const;
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSelected(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual void MouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
```

```

virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void ReSize(CPoint theNewLocation);
virtual void Select(void);
virtual void SetDragging(BOOLEAN isDraggable);
void SetGroupEnclosure(CSubview* theGroupEnclosure);
void SetGroupRegion(void);
virtual void SetFont(const FONT& theFont, BOOLEAN isUpdate = FALSE);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void Size(const CRect& theNewSize);
BOOLEAN SizeOwner(void);

```

Summary: CHWireFrame.h

```

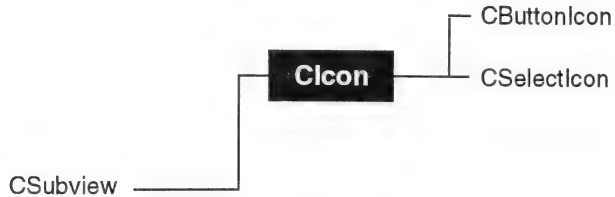
#include "PwrDef.h"
#include CWireFrame_i

class CHorizontalWireFrame : public CWireFrame
{
public:
    CHorizontalWireFrame(CView *theOwner, CSubview*
        theGroupEnclosure = NULL)
    :
        CWireFrame(theOwner, theGroupEnclosure)
    {
    }

    virtual void MouseMove(CPoint theLocation,short theButton = 0,
        BOOLEAN isShiftKey=FALSE,
        BOOLEAN isControlKey=FALSE)
    {
        // By using itsGrabPoint.V(), we pretend that all vertical
        // positions are identical to the original "grab" position:
        theLocation.V(theGrabPoint.V());
        CWireFrame::MouseMove(theLocation, theButton, isShiftKey,
            isControlKey);
    }
};

```

CIcon



Description

CIcon objects display icon resources inside CView objects. An icon resource is a drawing using pixels. An icon is a specific kind of view that is derived from CSubview. As a type of subview, icons can contain other objects nested within them—a nested text object, for example. CIcon is not an abstract class; it can be instantiated.

Heritage

Subclasses: CButtonIcon, CSelectIcon

Usage

When you create and initialize each CIcon object, it requires two resource IDs for the following icon states:

- enabledRID: the icon is enabled (can receive events)
- disabledRID: the icon is disabled (cannot receive events)

If you do not need both states, you can pass the same RID to both resource IDs. If a title is given, it is displayed below the icon.

To add functionality to icons, create derived classes that override the CIcon methods. Do not modify this base class. Available derived icons include CSelectIcon and CButtonIcon.

Consult Appendix B of the *XVT-Power++ Guide* for information on how to create icon resources for your particular platform.

Environment

Under some platforms, the icon drawing appears in the foreground color, as does its title. Open spaces in the drawing appear in the background color. You can set the background and foreground colors. Under other platforms, the icon's color is fixed as defined.

For portability, set the colors appropriately even if the information is not used.

Protected Data Members

int	itsCurrentRID;	current RID used for drawing
-----	----------------	------------------------------

Private Data Members

int	itsDisabledRID;	icon is disabled
int	itsEnabledRID;	icon is enabled to receive events
int	itsCenter;	origin to center icon
CRect	*itsIconSize;	internal size
PNT	itsTextOrigin;	origin used for the title

Public Methods

Constructor, Destructor, and Initializer Methods

```
CIcon(CSubview* theEnclosure, const CRect& theRegion);
```

A constructor. theEnclosure is a pointer to the subview that will contain the icon. theRegion is a coordinate location, local to the enclosure, that is used to place the icon.

```
Icon(const CIcon& theIcon);
```

A copy constructor that causes two icons to draw the same picture. As with all subviews, only the top-level view is copied. Any views nested within the icon are not copied.

```
CIcon& operator=(const CIcon& theIcon);
```

An assignment operator that causes two icons to draw the same picture. As with all subviews, only the top-level view is copied. Any views nested within the icon are not copied.

```
virtual ~CIcon(void);
```

The destructor. It cleans up after the icon and deletes any views nested within it.

```
virtual BOOLEAN IIcon(int theEnabledRID      = NULLicon,
                      int theDisabledRID     = NULLicon,
                      const CString theTitle = NULL,
                      BOOLEAN isVisible      = TRUE,
                      GLUETYPE theGlue      = NULL);
```

The initializer. It takes two integers, one called theEnabledRID, where RID stands for “resource ID” and one called

theDisabledRID. These two numbers represent the icon resources that have been created in accordance with the XVT Portability Toolkit's specifications for creating resources in a different platform. The first resource is displayed when the icon is in an enabled state and the second is displayed when the icon is in a disabled state. If you do not want to make a distinction between the display of enabled and disabled states, you can pass in the same number for both parameters. theTitle is the title of the icon, which is displayed underneath the icon and centered. Finally, the initializer takes a visibility state and a glue type.

Icon Utilities and Options

```
virtual void Invert(void);
```

Displays the icon resource for the icon's inverted mode. When created, icons are given three different resources to reflect the enabled, disabled, and inverted states. This method replaces the currently displayed resource with the resource for inverted mode.

Inherited Overridden Utilities

```
virtual void SetFont(const FONT& theFont,  
    BOOLEAN isUpdate = FALSE);
```

A method that does not affect the icon itself, but if the icon has a title, it sets the font of the title. This method has been overridden to ensure that it remains centered despite font changes. The isUpdate parameter indicates whether this SetFont message is simply an update message or whether it is a "real" SetFont message meaning that the icon should set its own font to the new font

```
virtual void SetTitle(const CString& theTitle);
```

Passes a new string to the icon to use as its title.

```
virtual void Draw(const CRect&  
    theClippingRegion);
```

Takes care of any drawing the icon must do. This method has been overridden to ensure that the icon is drawn properly. theClippingRegion is in global, window-relative coordinates.

```
virtual void Enable(void);
```

Enables the icon so that it can receive events and displays the icon resource for the icon's enabled mode.

```
virtual void Disable(void);
```

Disables the icon so that it cannot receive events and displays the picture designated by the disabled resource ID.

Sizing (Disabled for Icons)

```
virtual void Size(const CRect& theNewRegion);
```

A method that has been overridden to disable sizing for icons. An icon has one defined size based on its resource.

```
virtual void SetSizing(BOOLEAN isSizable);
```

A method that has been overridden to disable sizing for icons. An icon has one defined size based on its resource.

Protected Methods

```
void Refresh(void);
```

Refreshes the image of the icon on the screen. This is necessary when the icon changes state or its title changes. Refresh is called internally by the methods that change the icon's state.

Private Methods

```
void UpdateSize(void);
```

An internal method that updates the internal size representation of the icon when the font of the icon's title changes.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

```
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
```

```
virtual void Deactivate(void);
```

```
virtual void DoCommand(long theCommand, void* theData=NULL);
```

```
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
```

```
virtual void DoTimer(long theTimerId);
```



```

virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWindow* GetCWindow(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);

```

```

virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);

```

```

virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

Overrides

Derived classes can override the mouse functions.

Summary: Clcon.h

```

#ifndef CIcon_H
#define CIcon_H
#include "CSubview.h"

class CIcon : public CSubview
{
public:
    // Construct, Destruct, Initialize:
    CIcon(CSubview* theEnclosure, const CRect& theRegion);
    CIcon(const CIcon& theIcon);
    CIcon& operator=(const CIcon& theIcon);
    virtual ~CIcon(void);
    virtual BOOLEAN IIcon(int theEnabledRID = NULLicon,
        int theDisabledRID = NULLicon,
        const CString theTitle = NULL,
        BOOLEAN isVisible = TRUE,
        GLUTYPE theGlue = NULL);

    // Icon utilities & options:
    virtual void Invert(void);

```

```
// Inherited overridden utilities:
virtual void SetFont(const FONT& theFont,
    BOOLEAN isUpdate = FALSE);
virtual void SetTitle(const CString& theTitle);
virtual void Draw(const CRect& theClippingRegion);

virtual void Enable(void);
virtual void Disable(void);

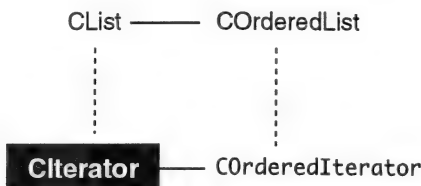
// sizing: disabled for icons
virtual void Size(const CRect& theNewRegion);
virtual void SetSizing(BOOLEAN isSizable);

protected:
    int itsCurrentRID; // Current RID used for drawing
    void Refresh(void);

private:
    int itsDisabledRID; // icon is disabled
    int itsEnabledRID; // icon is enabled to receive events
    int itsCenter; // origin to center icon
    CRect *itsIconSize; // internal size
    PNT itsTextOrigin; // origin used for the title
    void UpdateSize(void);
};

#endif CIcon_H
```

CIterator



Description

The CIterator class is used to iterate through the items in a CList. Iterators are guaranteed to iterate through every item, but no specific iterating order should be assumed. For more information, see CList.

Heritage

Superclass: none

Usage

Use the CIterator class to iterate over a list. Because of the use of void* you must use casting in at least two places when using CList under the current non-template implementation. Here is an example of use:

```

CList aList();
Foo* aFoo = new Foo(1,2,3);
aList.Insert((void*) aFoo); // Cast #1
Foo* fooItem;
CIterator doTo(aList);
while (fooItem = (aFoo*) doTo.Next()) // Cast #2
    fooItem->Goo();
  
```

Public Data Members

None.

Protected Data Members

CLink*	itsFirstLink;	a placeholder in a list for the first link
CLink*	itsCurrentLink;	a placeholder in the list for the link that is currently being iterated through

Private Data Members

None.

Public Methods

Constructor Methods

```
CIterator(const CList& theList);
```

A constructor. It takes a reference to a list, theList.

```
CIterator(const CList* theList);
```

A constructor. It takes a pointer to a list, theList.

Iterating Methods

```
virtual void* Next(void);
```

Returns the next item on the list each time it is called until it reaches the end of the list, in which case it returns NULL. You can iterate by starting a loop and continuing the loop as long as NEXT returns nonNULL values. Here is an example of use:

```
CList aList();
Foo* aFoo = new Foo(1,2,3);
aList.Insert((void*) aFoo); // Cast #1
...
Foo* fooItem;
CIterator doTo(aList);
while (fooItem = (aFoo*) doTo.Next()) // Cast #2
    fooItem->Goo();
```

```
virtual void* Previous(void);
```

Returns the preceding item on the list each time it is called until it reaches the top of the list, in which case it returns NULL.

```
virtual void* First(void);
```

Returns the first item on the list and sets the iterator to the first element. Calling Previous after using this method results in a return of NULL.

```
virtual void* Last(void);
```

Returns the last item on the list and sets the iterator to the last element. If you were to call Next after calling this method, the result would be NULL.

```
int StepForward(int theNumberOfSteps);
```

Steps forward in the iteration by theNumberOfSteps and returns an integer indicating how many steps it was actually able to

take. If theNumberOfSteps is 15 but there are only three items left on the list, this method will return a 3.

```
int StepBack(int theNumberOfSteps);
```

Steps backward in the iteration by theNumberOfSteps and returns an integer indicating how many steps it was actually able to take. If theNumberOfSteps is 15 but there are only three previous items on the list, this method will return a 3.

Summary: CLists.h

Note: CList and CIterator share the same header file. This file is shown in the discussion of each class.

```
#ifndef CLists_H
#define CLists_H

#include "xvt.h"
#include "PwrNames.h"
#include "CGlobal.h"
#include "CMem.h"

class CLink;
class CLink;
class CLinkHandle;

class CList
{
public:
    // Class Utility:
    CList(void);
    CList(const CList& theOtherList);
    CList& operator=(const CList& theOtherList);
    virtual ~CList(void);

    // List Operations:
    void Insert(void* theItem);
    BOOLEAN Remove(void* theItem);

    void Clear(void);

    int NumItems(void) const;
    BOOLEAN IsEmpty(void) const;

    int InList(void* theItem) const;

    CList& operator+=(const CList& theListToCatenate);
    friend CList operator+(const CList& theFirstList,
        const CList& theSecondList);

private:
    friend class CIterator; // class which iterates through the
                           // list
    friend class COorderedList;
    CLinkHandle* itsLinkHandle; // reference counting link handle

    CLink* GetFrontLink(void) const;
    void SetFrontLink(CLink* theLink);
```

```
    void DecrementCount(void);
    void IncrementCount(void);

    void DestructiveHandle();
    void InsertInternal(void* theItem);
    void InsertInternal(void* theItem, CLink* thePreviousLink);
};

class CIterator
{
    public:

        CIterator(const CList& theList);
        CIterator(const CList* theList);

        // Iterating methods:
        virtual void* Next(void);
        virtual void* Previous(void);

        virtual void* First(void);
        virtual void* Last(void);

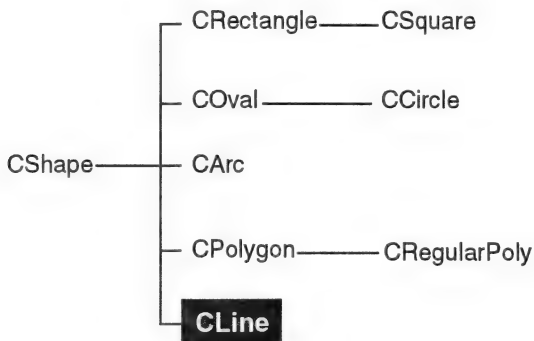
        int StepForward(int theNumberOfSteps);
        int StepBack(int theNumberOfSteps);

    protected:

        CLink* itsFirstLink;
        CLink* itsCurrentLink;
};

#endif CLists_H
```


CLine



Description

CLine objects paint a line inside a view.

Heritage

Superclass: CShape

Usage

Create and initialize a line. Like all CShape classes, this class inherits all properties of CSubview, such as stickiness, enclosure, and so on.

Environment

The line is drawn with the pen, which has settings for color, pattern, and width.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

BOOLEAN	itHasStartArrow;	whether the line has a starting arrow
BOOLEAN	itHasEndArrow;	whether the line has an ending arrow

CPoint	*itsStartPoint;	the point where the line starts
CPoint	*itsEndPoint;	the point where the line ends

Public Methods

Constructor, Destructor, and Initializer Methods

```
CLine (CSubview* theEnclosure, const CPoint&
theStartingPoint, const CPoint&
theEndingPoint);
```

A constructor. theEnclosure is a pointer to the subview that will contain the line. In addition, this method takes two points (CPoints), the point at which the line starts and the point at which it stops.

```
CLine(const CLine& theLine);
```

A copy constructor that creates a new CLine object with the same enclosure, color, visibility attributes, enabled/disabled attributes, environment, and so on as the original CLine object.

```
CLine& operator=(const CLine& theLine);
```

An assignment operator. It copies the attributes of the original CLine object, creating a new CLine object that has the same color, glue, environment, visibility state, and so on.

```
virtual ~CLine(void);
```

The destructor. It cleans up after the line and deletes any views nested within it.

```
BOOLEAN ILine(BOOLEAN hasBeginningArrow = FALSE,
BOOLEAN hasEndingArrow = FALSE,
BOOLEAN isVisible = TRUE,
long theGlue = NULLSTICKY);
```

The initializer. By default, the line has neither a beginning arrow nor an ending arrow, it is visible, and it uses the glue type NULLSTICKY. One or more of these defaults can be overridden.

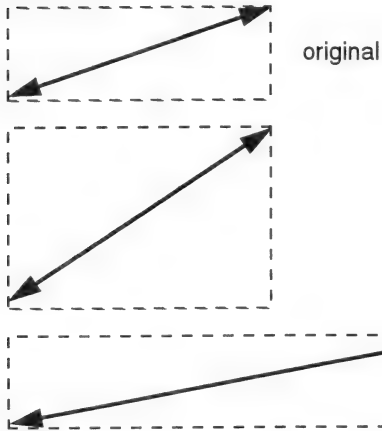
```
virtual void Draw(const CRect&
theClippingRegion);
```

Takes care of any drawing the line must do. theClippingRegion is the part of the line to be drawn, and it is in global, window-relative coordinates.

```
virtual void Size(const CRect& theNewSize);
```

Sizes the line according to the coordinates of theNewSize. The reset region could be in a different location and have completely

different dimensions—a new width or a new height. The coordinates of the region, theNewSize, are relative to the enclosure.



```
virtual void Size(const CPoint&
theStartingPoint, const CPoint&
theEndingPoint);
```

Resizes the line using theStartingPoint and theEndingPoint as the line's new starting and ending coordinates.

Utility Methods

```
virtual void SetArrows(BOOLEAN
hasBeginningArrow, BOOLEAN hasEndingArrow);
```

Allows you to set the beginning and ending arrows for the line, taking a Boolean value of TRUE or FALSE for each of its two parameters.

```
virtual BOOLEAN HasBeginningArrow(void);
```

Returns a Boolean value indicating whether the line has a beginning arrow.

```
virtual BOOLEAN HasEndingArrow(void);
```

Returns a Boolean value indicating whether the line has an ending arrow.

Protected Methods

None.

Private Methods

None.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);  
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);  
virtual CUnits* GetUnits(void) const;  
virtual void SetUnits (CUnits* theCoordinateUnits);  
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);  
virtual void Deactivate(void);  
virtual void Disable(void);  
virtual void DoCommand(Long theCommand, void* theData=NULL);  
virtual void DoPrintDraw(const CRect& theClippingRegion);  
virtual void DoTimer(long theTimerId);  
virtual void DoUser(long theUserId, void* theData);  
virtual void Draw(void);  
virtual void Enable(void);  
virtual CView* FindHitView(const CPoint& theLocation) const;  
virtual CRect GetClippedFrame(void) const;  
virtual long GetCommand(void) const;  
virtual CWindow* GetCWindow(void) const;  
virtual long GetDoubleCommand(void) const;  
virtual CSubview* GetEnclosure(void);  
virtual const CEnvironment* GetEnvironment(void) const;  
virtual CRect GetFrame(void) const;  
virtual CRect GetGlobalFrame(void) const;  
virtual CPoint GetGlobalOrigin(void) const;  
virtual GLUETYPE GetGlue(void) const;  
virtual int GetId(void) const;  
virtual CRect GetLocalFrame(void) const;  
virtual CPoint GetOrigin(void) const;  
virtual const CString GetTitle(void) const;
```

```
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDoubleClick(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;

```

```

virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CShape

None.

Summary: CLine.h

```

#ifndef CLine_H
#define CLine_H
#include "CShape.h"
class CLine : public CShape
{
public:
    // Construct, destruct, init
    CLine(CSubview* theEnclosure, const CPoint& theStartingPoint,
          const CPoint& theEndingPoint);
    CLine(const CLine& theLine);
    CLine& operator=(const CLine& theLine);
    virtual ~CLine(void);

    BOOLEAN ILine(BOOLEAN hasBeginningArrow = FALSE,
                  BOOLEAN hasEndingArrow = FALSE,
                  BOOLEAN isVisible = TRUE,
                  long theGlue = NULLSTICKY);

    virtual void Draw(const CRect& theClippingRegion);
    virtual void Size(const CRect& theNewSize);
    virtual void Size(const CPoint& theStartingPoint,
                      const CPoint& theEndingPoint);

    // util:
    virtual void SetArrows(BOOLEAN hasBeginningArrow,
                           BOOLEAN hasEndingArrow);
    virtual BOOLEAN HasBeginningArrow(void);
    virtual BOOLEAN HasEndingArrow(void);

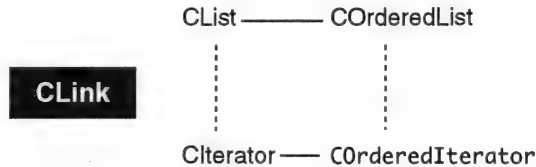
protected:

private:
    BOOLEAN itHasStartArrow;    // wants a starting arrow
    BOOLEAN itHasEndArrow;     // wants an ending arrow
    CPoint *itsStartPoint;     // start drawing line here
    CPoint *itsEndPoint;       // end drawing line here
};

#endif CLine_H

```

CLink



Description

CLink is a CList helper class. This private class is a data structure representing each of the links in the chain of a list. CLink objects function as containers of individual objects in a list.

Heritage

Superclass: none

Usage

This is a private class that is used solely for CList.

Data Members

None.

Private Data Members

CLink*	itsNextLink;	returns a pointer to the next link in the list
CLink*	itsPreviousLink;	returns a pointer to the previous link in the list
void*	itsItem;	returns a pointer to the item in the link

Public Methods

The CLink data structure contains three pointers: to the previous link, to the next link, and to the item of a link. The following methods provide ways to get and set these pointers.

```
void SetNext(CLink* theNextLink);
```

Given a pointer to a link (theNextLink), sets it as the next link in the list.

```
CLink* GetNext(void) const;
```


Returns a pointer to the next link in the list.

```
void SetPrevious(CLink* thePrevious);
```

Given a pointer to a link (thePrevious), sets it as the previous link in the list.

```
CLink* GetPrevious(void) const;
```

Returns a pointer to the previous link in the list.

```
void* GetItem(void) const;
```

Returns a pointer to the link's item.

```
void SetItem(void* theItem);
```

Given a pointer to an item, sets this item as the link's item.

Private Methods

```
CLink(void);
```

A constructor, which creates an empty link.

```
CLink (CLink *theNextLink, CLink*  
thePreviousLink, void *theItem);
```

A constructor. It takes pointers to the next link and the previous link in the list and to the item in the link.

Summary: CLink.h

```
#ifndef CLink_H
#define CLink_H

#include "PwrNames.h"
#include "CGlobal.h"

class CLink
{
    friend class CList;
    friend class CIterator;

public:
    void SetNext(CLink* theNextLink);
    CLink* GetNext(void) const;

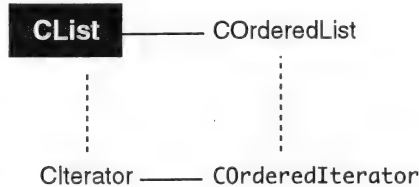
    void SetPrevious(CLink* thePrevious);
    CLink* GetPrevious(void) const;

    void* GetItem(void) const;
    void SetItem(void* theItem);

private:
    CLink(void);
    CLink (CLink *theNextLink, CLink* thePreviousLink,
           void *theItem);
```

```
        CLink* itsNextLink;  
        CLink* itsPreviousLink;  
        void* itsItem;  
};  
#endif CLink_H
```

CList



Description

CList is a container class that can store references to any generic object. Since templates are not supported by every C++ compiler, objects are inserted and removed via void pointers. Thus, **CList** stores objects by their address. It does not store “copies” of objects. The drawback to void pointers is that you have to remember the type of the item that you insert into a list. It is recommended that you store only one type of item in each list so that you can cast it in and cast it out safely.

A **CList** object is composed of links, which are data structures that function as containers for individual items in a list. The **CLink** data structure contains three pointers: to the previous link, to the next link, and to the item of data in the link.

You cannot assume any order for storage of items or any means of storage (array, linked list, and so on). Objects can be inserted twice, creating multiple references to the same object within the list. Removing an item, however, removes only one instance of a multiply-stored object.

The **CIterator** class is used to iterate through the items in the list. Iterators are guaranteed to iterate through every item, but no specific iterating order should be assumed.

See Also: **CLink**

Heritage

Superclass: none

Usage

Create a list and then insert and remove items. Use the **CIterator** class to iterate over a list. Because of the use of **void***, you must use

casting in at least two places when using CList under the current non-template implementation.

Here is an example of use:

```
CList aList();
Foo* aFoo = new Foo(1,2,3);
aList.Insert((void*) aFoo); // Cast #1
Foo* fooItem;
CIterator doTo(aList);
while (fooItem = (aFoo*) doTo.Next()) // Cast #2
    fooItem->Goo();
```

Public Data Members

None.

Protected Data Members

None.

Private Data Members

None.

Friends

friend class	CIterator;	class that iterates through the list
friend class	COrderedList;	an ordered list
CLinkHandle	*itsLinkHandle;	since lists are reference counted, this handle contains the implementation of the reference count

Public Methods

Class Utility

CList(void);

A constructor, which creates an empty list.

CList(const CList& theOtherList);

A copy constructor, which copies all of the attributes of theOtherList, including any pointers it contains. It does not copy the objects to which the list is pointing. Thus, the contents of both lists will be identical.

CList& operator=(const CList& theOtherList);

An assignment operator, which copies all of the attributes of theOtherList, including any pointers it contains. It does not copy the objects to which the list is pointing.

```
virtual ~CList(void);
```

The destructor, which cleans up after the list but does not delete the objects to which the list is pointing.

List Operations

```
void Insert(void* theItem);
```

Inserts an item into a list. theItem is a void pointer to a generic object.

```
BOOLEAN Remove(void* theItem);
```

Removes an item from the list. That is, it removes a void pointer (theItem) to a generic object. It returns a Boolean value of TRUE if it succeeds. If an item is multiply stored in the list, this method removes only the first occurrence of this item that it encounters.

```
void Clear(void);
```

Empties the list.

```
int NumItems(void) const;
```

```
BOOLEAN IsEmpty(void) const;
```

Returns a Boolean value of TRUE if the list is empty and FALSE if it is not.

```
int InList(void* theItem) const;
```

Given a pointer (theItem), searches for this pointer on a list. It returns an integer indicating how many times this item was found on the list.

```
CList& operator+=(const CList&  
theListToCatenate);
```

Concatenates two lists. Specifically, it concatenates theListToCatenate with the current list.

```
friend CList operator+(const CList&  
theFirstList, const CList& theSecondList);
```

Concatenates theFirstList with theSecondList. Unlike the preceding method, operator+==, this method does not actually

change the lists. Instead, it makes a temporary copy of the list created when the change is made.

Private Methods

CLink* GetFrontLink(void) const;

Returns the front link on a list, that is, the list's first item.

void SetFrontLink(CLink* theLink);

Sets theLink as the front link on a list, that is, the list's first item.

void DecrementCount(void);

When a link is removed from a list, this internal method decrements the internal count.

void IncrementCount(void);

When a link is added to a list, this internal method increments the internal count.

void DestructiveHandle();

A method that is called whenever a list is about to change to ensure that the list is not being multiply referenced.

void InsertInternal(void* theItem);

An internal method that is used to actually insert an item into a list.

void InsertInternal(void* theItem, CLink* thePreviousLink);

An internal method that is used to actually insert an item into a list. It takes not only theItem but also thePreviousLink on the list.

Summary: CLists.h

```
#ifndef CLists_H
#define CLists_H

#include "xvt.h"
#include "PwrNames.h"
#include "CGlobal.h"
#include "CMem.h"

class CLink;
class CLinkHandle;

class CList
{
public:
```

```

// Class Utility:
CList(void);
CList(const CList& theOtherList);
CList& operator=(const CList& theOtherList);
virtual ~CList(void);

// List Operations:
void Insert(void* theItem);
BOOLEAN Remove(void* theItem);

void Clear(void);

int NumItems(void) const;
BOOLEAN IsEmpty(void) const;

int InList(void* theItem) const;
CList& operator+=(const CList& theListToCatenate);
friend CList operator+(const CList& theFirstList,
                      const CList& theSecondList);

private:

    friend class CIterator;           // class which iterates through the list
    friend class COrderedList;
    CLinkHandle* itsLinkHandle;      // reference counting link handle

    CLink* GetFrontLink(void) const;
    void SetFrontLink(CLink* theLink);

    void DecrementCount(void);
    void IncrementCount(void);

    void DestructiveHandle();
    void InsertInternal(void* theItem);
    void InsertInternal(void* theItem, CLink* thePreviousLink);
};

class CIterator
{
public:
    CIterator(const CList& theList);
    CIterator(const CList* theList);

    // Iterating methods:
    virtual void* Next(void);
    virtual void* Previous(void);

    virtual void* First(void);
    virtual void* Last(void);

    int StepForward(int theNumberOfSteps);
    int StepBack(int theNumberOfSteps);

protected:
    CLink* itsFirstLink;
    CLink* itsCurrentLink;
};

#endif CLists_H

```

CListBox

CSubview — CVirtualFrame — CScroller — **CListBox**

Description

CListBox objects display a box containing a list of selectable text items. The box can have horizontal and vertical scrollbars. To select an item inside the list box, the user clicks the mouse on it.

Heritage

Superclass: CScroller

Usage

Create an instance of this class, initialize it, and register it with an enclosure. Items can be dynamically added, removed, activated, or disabled. The items are numbered, starting at 0, from the top down.

All items passed to the object to be inserted into the list are assumed to be pointers to character strings. That is, the initializing routine takes an ordered list of CStrings.

When the list box is sized, or when a font is changed, the final size of the list box's viewing rectangle can optionally be adjusted so that an integral number of lines can be viewed. This option assures that a line will not be cut halfway through or halfway clipped at the bottom of the list.

Environment

The border of the list box is drawn with the pen; its interior is painted with the brush. The text items are drawn in the background color, but, when selected, are drawn in the foreground color. The environment properties of the scrollbars are system-defined. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width. Finally, you can set the foreground and background colors.

Public Data Members

None.

Protected Data Members

CIdOrderedList *itsItems;	list that stores the actual text strings
---------------------------	--

Private Data Members

CFixedGrid	*itsGrid;	grid used to display items
BOOLEAN	itIsDragging;	whether the mouse is being dragged
UNITS	suggestedHeight;	used for sizing
BOOLEAN	itIsHit;	keeps track of mouse event state

Public Methods

Constructor, Destructor, and Initializer Methods

CListBox(CSubview* theEnclosure, const CRect& theRegion);

A constructor. theEnclosure is a pointer to the subview that will contain the list box. theRegion is a coordinate location, local to the enclosure, that is used to place the list box.

CListBox(const CListBox& theListBox);

A copy constructor that duplicates the attributes of a list box. The items inside the box are currently not being copied.

CListBox& operator=(const CListBox& theListBox);

An assignment operator that duplicates the attributes of a list box, as well as the items inside the list box. The items inside the box are currently not being copied.

virtual ~CListBox(void);

The destructor, which cleans up after the listbox and deletes any items contained within it.

BOOLEAN IListBox(const COrderedList* theItems = NULL, BOOLEAN hasHorizontalScrollBar = FALSE, BOOLEAN hasVerticalScrollBar = TRUE, BOOLEAN isThumbTracking = TRUE, long theSingleClickCommand = NULLcmd, long theDoubleClickCommand = NULLcmd, BOOLEAN isVisible = TRUE, long theGlue = NULLSTICKY);

The initializer. It takes an ordered list of CString items that are to be inserted into the list box (as shown in the example). It

allows you to specify whether the list box's scroller has a horizontal scrollbar, whether it has a vertical scrollbar, and whether it dynamically tracks the scrollbar's thumb. `theSingleClickCommand` specifies the command that is generated upon a single mouse click; `theDoubleClickCommand` specifies the command that is generated upon a double mouse click. Like most initializers for classes in the `CSubview` hierarchy, this initializer takes a visibility state and a glue type.

Example:

```
CListBox* aListBox = new CListBox(enc, aRect);
COrderedList items;

CString str1("This is line 1");
items.Insert(&str1,0);

CString str2("This is line 2");
items.Insert(&str2,1);

...

CString strn("This is line n");
items.Insert(&strn,n-1);

aListBox->IListBox(&items);
```

Text Item Utility

virtual void InsertLine(const CString& theItem, int thePosition);

Inserts a line of text by giving the list box a new string item and a position on the list in which to insert this string. `thePosition` starts at zero (0) with the topmost item in the list.

virtual void RemoveLine(int thePosition);

Given the number of a position on the list, removes the string item that occupies this position. `thePosition` starts at zero (0) with the topmost item in the list.

virtual int GetNumberOfLines(void) const;

Returns the number of lines in the list box.

virtual int GetSelectedLine(void) const;

Returns the number of the line in the listbox that is selected. It returns minus one (-1) if no line is selected.

virtual void SelectLine(int theLine);

Given the number of a line (`theLine`) in the list box, selects that line.

```
virtual void DeselectLine(void);
```

Deselects the currently selected line.

```
virtual CString GetLine (int thePosition)  
const;
```

Takes an integer (thePosition), which is a line number, and returns the string that is displayed for that line in the list box. Line numbers start at zero (0).

```
virtual void ScrollUntilVisible(int  
thePosition);
```

Scrolls the list box so that the given line position (thePosition) is displayed. Line numbers start at zero (0).

List Box Sizing

```
virtual BOOLEAN IsHeightSelfAdjusted(void);
```

Returns a Boolean value indicating whether the list box can resize itself so that an integral number of lines are shown.

```
virtual void SelfAdjustHeight(BOOLEAN  
isHeightSelfAdjusted);
```

By default, the programmer specifies the height of a list box. Thus, if you specify that you want the list box to be 10 pixels tall, then it is 10 pixels tall. Sometimes the result of this size specification is that a certain line inside the list box is only partially displayed so that the user must scroll up and down to see it. SelfAdjustHeight takes a Boolean value indicating whether a list box can adjust its own height so that an integral number of lines are shown.

Inherited Utility Methods

```
virtual void Key(int theKey, BOOLEAN  
isShiftKey, BOOLEAN isControlKey);
```

Sends a keyboard event to a list box. theKey is the ASCII value for the character key that was pressed; IsShiftKey and IsControlKey indicate whether the SHIFT or CONTROL key was pressed in conjunction with the character key.

```
virtual void VScroll(SCROLL_CONTROL  
theEventType, UNITS theThumbPosition);
```

A method called by the list box's scrollbars to propagate scrollbar events. An event propagates from the scrollbar to the CScroller object. theEventType designates the kind of scroll: line up, line down, page up, or page down. theThumbPosition is the numerical value for placement of the thumb when either

thumb repositioning or dynamic thumb tracking is used. This method responds to the scrollbar event by scrolling the views contained inside the list box.

virtual void Size(const CRect& theNewSize);

Sizes the list box according to the coordinates of theNewSize. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, theNewSize, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated. If the height of the list box is self-adjusting, theNewSize may be modified to include an integral number of lines.

virtual void DoSize(const CRect& theNewSize);

Sizes the list box according to the coordinates of theNewSize. Calling DoSize for list boxes is equivalent to calling Size since they both take care of updating the locations of the list box's contents.

void DoDraw(const CRect& theClippingRegion);

The list box refreshes (draws) itself and notifies each of its subviews to do the same. theClippingRegion is the part of the list box that needs to be refreshed and is in global, window-relative coordinates; if it is set to NULL, the entire list box is drawn. This method is automatically called after an E_UPDATE event.

virtual void DoDraw(void);

The listbox draws itself and notifies each of its subviews to do the same. By default the clipping region is set to the entire region that the listbox occupies. If you do not want to calculate a clipping region and just want to draw the entire listbox, then you can call this method without passing anything.

virtual void SetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);

Takes a font and sets the list box's environment to have that font. It recursively passes theNewFont to all of its string items (subviews). The isUpdate parameter indicates whether this SetFont message is simply an update message or whether it is a "real" SetFont message meaning that the list box should set its own font to the new font.

Selecting Routines

All of the following methods have been overridden to implement some selection and deselection on different mouse events when the user clicks on the list box. The mouse events are passed directly to the view at the location where the event occurs. Each mouse method has a `theButton` argument that specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right).

```
virtual CView* FindEventTarget(const CPoint&
theLocation) const;
```

A method that is called to find the target of a mouse event within a subview. This virtual method is defined to return the deepest subview at the global coordinates of `theLocation`. This method can be overridden by a subview in order to trap all events regardless of the subviews it contains or to change the event targeting algorithm. For example, a list box contains many subviews (text items). When you click on a text item inside a list box, the event does not go to the `CText` view. Instead, the list box overrides `FindEventTarget` and traps the event by returning this (itself) as the target for the mouse events received. Then, it treats the mouse events as necessary, without sending them on to its subviews.

```
virtual void MouseDown(CPoint theLocation,
                        short theButton      = 0,
                        BOOLEAN isShiftKey    = FALSE,
                        BOOLEAN isControlKey  = FALSE);
```

When the user presses down a mouse button over a list box containing local coordinate `theLocation`, the list box receives and handles this event. If `theLocation` matches an item in the list box, this item becomes selected.

```
virtual void MouseUp(CPoint theLocation,
                     short theButton      = 0,
                     BOOLEAN isShiftKey    = FALSE,
                     BOOLEAN isControlKey  = FALSE);
```

When the user releases a mouse button over a list box containing local coordinate `theLocation`, the list box receives and handles this event. This event is not necessarily preceded by a `MouseDown` or `MouseMove` event because those button actions may have taken place outside the icon.

```
virtual void MouseMove(CPoint theLocation,
                       short theButton      = 0,
                       BOOLEAN isShiftKey    = FALSE,
                       BOOLEAN isControlKey  = FALSE);
```

When the user moves the mouse over a list box containing local coordinate `theLocation`, the list box receives and handles this event, whether the mouse button is down or not.

```
virtual void MouseDouble(CPoint theLocation,  
                          short theButton    = 0,  
                          BOOLEAN isShiftKey = FALSE,  
                          BOOLEAN isControlKey = FALSE);
```

When the user double clicks a mouse button over a list box containing local coordinate `theLocation`, the list box receives and handles this event. A double click command is generated.

Inherited Methods

```
virtual void DoSetFont(const FONT& theNewFont,  
                       BOOLEAN isUpdate = FALSE);
```

Takes a font and sets the list box's environment to have that font. It recursively passes the `theNewFont` to all subviews of the list box. The `isUpdate` parameter indicates whether this `SetFont` message is simply an update message or whether it is a "real" `SetFont` message meaning that the list box should set its own font to the new font.

```
virtual void SetFont(const FONT& theNewFont,  
                     BOOLEAN isUpdate = FALSE);
```

Takes a font (`theNewFont`) and sets the list box's environment to have that font. The `isUpdate` parameter indicates whether this `SetFont` message is simply an update message or whether it is a "real" `SetFont` message meaning that the list box should set its own font to the new font.

```
virtual void DoSetSizing(BOOLEAN isSizable);
```

Takes a Boolean value that sets the sizing of a list box to `TRUE` so that the scroller can be sized or to `FALSE` so that the list box cannot be sized. It then recursively sets the sizing for the list box's child views.

```
virtual void DoSetDragging(BOOLEAN  
                           isDraggable);
```

Takes a Boolean value that sets the dragging of a scroller to `TRUE` so that the list box can be dragged with the mouse or to `FALSE` so that the list box cannot be dragged. It then recursively set the dragging for the list box's child views.

```
virtual void SetGlue(GLUETYPE theGlue);
```

Sets the glue type of the list box and then recursively sets the glue of the its subviews to that type.

Private Methods

```
void SizeLines(UNITS theWidth, UNITS theHeight);
```

An internal method that is called to size the lines in a list box after a font change or other kind of change.

```
void ResetDimensions(void);
```

An internal method that resets the list box's dimensions in accordance with font changes.

```
void ScrollLines(CPoint theLocation);
```

An internal method that is called to scroll the lines in a list box after a font change or other kind of change.

```
CView* GetSelectedItem(int theIncrement);
```

An internal method that is used for scrolling. It returns the item above or below the selected item.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Enable(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
```

```
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Hide(void);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
```


From CSubview

```

virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoShow(void);
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation,CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CVirtualFrame

```

virtual void Draw(const CRect& theClippingRegion);

```

```

virtual void EnlargeToFit(const CRect& theRegionToInclude);
virtual CPoint GetGlobalOrigin(void) const;
CPoint GetScrollingOrigin(void);
virtual CRect GetVirtualFrame(void) const;
virtual void PostSize(void);
virtual void PreSize(void);
void SetScrollingOrigin(const CPoint& theNewOrigin);
virtual void SetVirtualFrame(unsigned int theNewWidth, unsigned int theNewHeight);

```

From CScroller

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int horizontalChange, int verticalChange);
CRect GetFrame(void) const;
virtual int GetHLineIncrement(void) const;
virtual int GetVLineIncrement(void) const;
virtual int GetHPageIncrement(void) const;
virtual int GetVPageIncrement(void) const;
void Glue(void);
virtual void HScroll(SCROLL_CONTROL theEventType, int theThumbPosition);
virtual BOOLEAN IsThumbtracking(void) const;
virtual void ScrollViews(const CPoint& theNewOrigin);
virtual void SetHIncrements(int theLineIncrement, int thePageIncrement);
virtual void SetHScrollRange(int theLeft, int theRight, int thePosition);
virtual void SetThumbtracking(BOOLEAN isThumbTracking);
virtual void SetVIncrements(int theLineIncrement, int thePageIncrement);
virtual void SetVScrollRange(int theTop, int theBottom, int thePosition);
virtual void ShrinkToFit(void);

```

Summary: CListBox.h

```

#ifndef CListBox_H
#define CListBox_H

#include "PwrDef.h"
#include CScroller_i
#include COrderedList_i

class CFixedGrid;
class CText;

class CListBox : public CScroller
{
public:

```

```

// Create, Init, Destruct
CListBox(CSubview* theEnclosure, const CRect& theRegion);
CListBox(const CListBox& theListBox);
CListBox& operator=(const CListBox& theListBox);
virtual ~CListBox(void);

BOOLEAN IListBox(const COrderedList* theItems = NULL,
    BOOLEAN hasHorizontalScrollBar = FALSE,
    BOOLEAN hasVerticalScrollBar = TRUE,
    BOOLEAN isThumbTracking = TRUE,
    long theSingleClickCommand = NULLcmd,
    long theDoubleClickCommand = NULLcmd,
    BOOLEAN isVisible = TRUE,
    long theGlue = NULLSTICKY);

// Text Item Utility:
virtual void InsertLine(const CString& theItem, int thePosition);
virtual void RemoveLine(int thePosition);

virtual int GetNumberOfLines(void) const;
virtual int GetSelectedLine(void) const;

virtual void SelectLine(int theLine, BOOLEAN isSelected);
virtual void DeselectLine(void);

virtual CString GetLine(int thePosition) const;
virtual void ScrollUntilVisible(int thePosition);

// List Box Sizing
virtual BOOLEAN IsHeightSelfAdjusted(void);
virtual void SelfAdjustHeight(BOOLEAN isHeightSelfAdjusted);

// Inherited Utility:
virtual void Key(int theKey, BOOLEAN isShiftKey,
    BOOLEAN isControlKey);
virtual void VScroll(SCROLL_CONTROL theEventType,
    UNITS theThumbPosition);
virtual void Size(const CRect& theNewSize);
virtual void DoSize(const CRect& theNewSize);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoDraw(void);

// Selecting routines:
virtual CView* FindEventTarget(const CPoint& theLocation) const;

virtual void MouseDouble(CPoint theLocation, short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);

//Inherited:
virtual void DoSetFont(const FONT& theNewFont,
    BOOLEAN isUpdate = FALSE);
virtual void SetFont(const FONT& theNewFont,
    BOOLEAN isUpdate = FALSE);

```

```
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoSetDragging(BOOLEAN isDraggable);

virtual void SetGlue(GLUETYPE theGlue);

protected:
    CIdOrderedList *itsItems;    // List to store items
private:
    CFixedGrid *itsGrid;         // Grid used to display items
    BOOLEAN itIsDragging;        // Is the mouse being dragged?
    UNITS suggestedHeight;       // Used for sizing
    BOOLEAN itIsHit;             // Keep track of mouse event state.

    void SizeLines(UNITS theWidth, UNITS theHeight);
    void ResetDimensions(void);
    void ScrollLines(CPoint theLocation);
    CView* GetSelectedItem(int theIncrement);
};

#endif CListBox_H
```

CMem

CMem

CError

CGlue

CDesktop

CEnvironment

CGlobal

CSwitchBoard

CStartup

CGlobalClassLib

CGlobalUser

CResourceMgr

CPrintMgr

Description

Currently **CMem** is a utility file, not a class. It overloads the global New and global Delete operators so that memory is allocated through the XVT Portability Toolkit free and malloc functions. This is especially important for DOS or Windows because the XVT Portability Toolkit greatly improves memory usage. In addition, this file provides debugging utility functions that are enabled when the MEM_DEBUG option is used.

The debugging utility counts the number of references to newly created and deleted objects. At any point, the user can see how many memory objects have been deleted. In addition, **CMem** keeps a list of the places, the file and the line, where the memory was allocated. When the user finishes running the program, he/she can print this list.

CMem will either become a memory management class or be incorporated into one of the top-level classes from which most of the classes inherit so that it can manage the creation and allocation of objects. Although this file is currently distributed with XVT-Power++ and some of the classes use it internally, XVT-Power++ will not officially support memory management or debugging utilities until a later release.

Heritage

Superclass: none

Usage

Include this file as the first **include** file in your **.cxx** file. This is done automatically if your first include is an XVT-Power++ **include** file.

Public Methods

```
void far* operator new(size_t sz, char* file,
    int line);
```

Allocates memory and stores the file and line number of the object for which it was allocated.

```
void operator delete(void far *p);
```

Deallocates memory and removes it from the memory list.

```
void DumpMemory(void);
```

Prints out into stdout the list of all objects for which memory has been allocated and not yet deallocated. Each object is designated by file and line number.

```
void CleanMemory(void);
```

Deallocates all memory allocated. This method, if called, must be followed by terminating the application.

Summary: CMem.h

```
#ifndef MEM_H
#define MEM_H
#include "xvt.h"
#ifdef MEM_DEBUG

    // Allocate memory and store the file and line from where it was
    // allocated.
void far* operator new(size_t sz, char* file, int line);

    // Deallocate memory and remove from memory list.
void operator delete(void far *p);

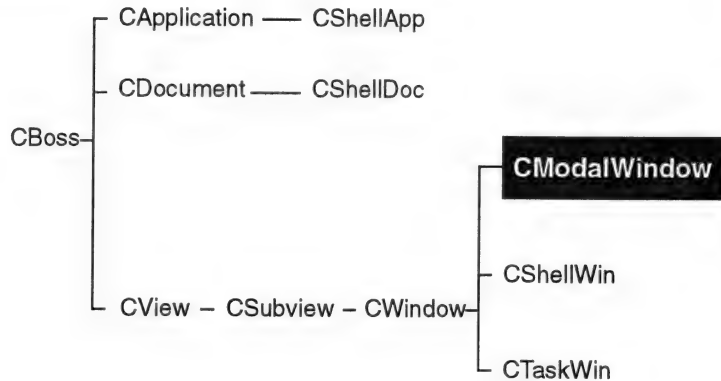
    // Dumps the line number and file of all instances memory allocations
    // lacking corresponding memory deallocations.
void DumpMemory(void);

    // Deallocates all memory allocated.
    // This function, if called, must be followed by terminating the
    // application.
void CleanMemory(void);

void * operator new(size_t sz, char* file, int line);
#else // For non-debugging applications:
```

```
inline void * operator new(size_t sz)
{
    void * temp = xvt_fmalloc(sz);
    return temp;
}
inline void operator delete(void far *p)
{
    if(p)
        xvt_ffree((char far*)p);
}
#endif
```

CModalWindow



Description

CModalWindow is a class that gives modal behavior to a window. When a modal window is opened, it takes over the screen, disabling all other windows so that nothing else can happen while it is open. For example, on the Macintosh, when a modal window appears, all other operations are disabled. Most items on the menubar are greyed-out, and all of the windows in the background become disabled. The modal window appears in the middle of the screen, and you cannot even move it or size it until you press the necessary button(s) on the window or respond to it in another way that is indicated. Then everything returns to normal. Modal windows are used when the program needs a certain item of information or a commitment from the user before it can continue. Regular windows, of course, do not freeze everything on the screen and can go into the background when another window is brought to the front.

The advantage of CModalWindow is that its objects behave like modal dialog windows (see CDialog) but do not require the use of URL resources. They inherit all the properties of CWindow, including the ability to nest a wide variety of XVT-Power++ views. Dialog windows, on the other hand, cannot nest XVT-Power++ views; they must contain only objects that are defined as controls in a URL resource file. You cannot add anything to a dialog window; you can only instantiate it and interact with it. In short, CModalWindow gives you the modal behavior and the platform-specific look and feel of a CModalDialog object without the restraints imposed on a CModalDialog object.

Heritage

Superclass: CWindow

Usage

First define a layout for a modal window, along with its window attributes and window type, just as you would for a regular CWindow. Then instantiate the object. Note that the window will not behave modally until you call CDesktop::DoModal and pass it a pointer to that window.

For hands-on practice in creating and using modal windows and dialog windows, consult the “Ask” example that is provided on the XVT-Power++ distribution disk.

Public Data Members

None.

Public Methods

```
CModalWindow(CDocument* theDocument, const CRect& theRegion,
             const CString& theTitle    = NULLString,
             long theWindowAttributes   = NATIVEDialogAttr,
             WIN_TYPE theWindowType     = NativeDialogType,
             int theMenuBarId           = MENU_BAR_RID);
```

A constructor. theDocument is a pointer to the window’s document. The coordinates of theRegion are screen-relative coordinates, and their meaning depends upon the platform. Some platforms honor these coordinates, and others do not; however, the size of the CRect is always important. theWindowAttributes takes a value from a set of XVT Portability Toolkit-provided attributes that you can give to windows. You can OR together the appropriate flags into an attribute value. The possible flags for theWindowAttributes are shown in the following table. The XVT Portability Toolkit supplies other, platform-specific flags.

WSF_NONE	no flags set
WSF_SIZE	is user-sizeable
WSF_CLOSE	is user_closeable
WSF_HSCROLL	has horizontal scrollbar outside client area
WSF_VSCROLL	has vertical scrollbar outside client area
WSF_DECORATED	all of above four flags are set
WSF_INVISIBLE	is initially invisible

WSF_DISABLED	is initially disabled
WSF_ICONIZABLE	is iconizable
WSF_ICONIZED	is initially iconized
WSF_NO_MENUBAR	has no menubar of its own
WSF_MAXIMIZED	initially maximized

theWindowtype specifies the type of the window. For a listing of the possible types, refer to the table at the beginning of the section on CWindow. Finally, theMenuBarId is the resource ID number of the menubar, which must be a valid URL-based resource.

Protected Methods

None.

Private Methods

WIN_TYPE GetInitialType(WIN_TYPE theType);

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual CRect GetFrame(void) const;
```

```
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;

```

```

virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);

```

From CWindow

```

virtual void ChangeFont(FONT theNewFont, FONT_PART thePart);
virtual BOOLEAN Close(void);
virtual void Disable(void);
virtual void DoActivateWindow(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoControl(int theControlID, CONTROL_INFO theControlInfo);
virtual void DoDeactivateWindow(void);
virtual void DoHScroll(SCROLL_CONTROL theEvent, short thePos);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void DoVScroll(SCROLL_CONTROL theEvent, short thePos);
virtual void Draw(const CRect& theClippingRegion);
virtual void Enable(void);
virtual long GetAttributes(void) const;
virtual CRect GetClippedFrame(void) const;
virtual CDocument* GetDocument(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual NWinScrollBar* GetHScrollBar(void) const;
virtual int GetMenuBarId(void) const;
virtual NWinScrollBar* GetVScrollBar(void) const;
virtual WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual BOOLEAN IsBackgroundDrawn(void) const;
virtual BOOLEAN IsClosable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void SetBackgroundDrawing(BOOLEAN isBackgroundDrawn);

```

```

virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetSelectedView(CView *theSubview);
virtual void SetTitle(const CString& theNewTitle);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void SizeWindow(int theWidth, int theHeight);
virtual void UpdateMenus(void);
virtual void UpdateUnits(CUnits* theUnits);

```

Summary:: CModalWindow.h

```

#ifndef CModalWindow_H
#define CModalWindow_H

#include "PwrNames.h"
#include "CWindow.h"

class CModalWindow : public CWindow
{
public:
    CModalWindow(CDocument* theDocument,
                 const CRect& theRegion,
                 const CString& = NULLString,
                 long theWindowAttributes = WSF_NONE,
                 WIN_TYPE theWindowType = W_DOC,
                 int theMenuBarId = MENU_BAR_RID);

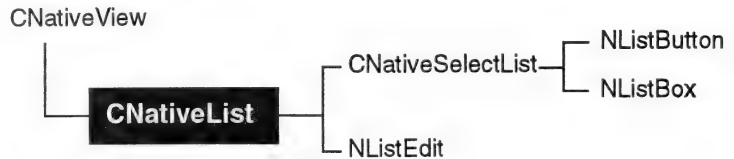
private:
    WIN_TYPE GetInitialType(WIN_TYPE theType);

#ifdef WSWIN
    BOOLEAN itsAttrSetting;
#endif
};

#endif CModalWindow_

```

CNativeList



Description

CNativeList is an abstract class from which several native list control classes are derived. Native list controls are objects such as list buttons, list boxes, and list edits that have the look and feel of the native window manager.

Heritage

Superclass: CNativeView

Subclasses: CNativeSelectList, NListEdit

Usage

Override this base class with classes that are specific to each XVT Portability Toolkit list control.

Environment

There are no environment settings for classes that inherit from CNativeView. The drawing properties of native views are system-defined.

Data Members

None.

Public Methods

Methods for Inserting and Removing Strings and Clearing the List

```
virtual BOOLEAN Insert(const CStringList&  
theStringList, int thePosition=-1);
```

Takes a list of strings and inserts it into the native list, starting at the specified position. Note that the default value of thePosition is -1, which means that the insertion starts at the end of any other list of strings already contained in the native list; that is, it appends the given list to the existing list. This

method returns a Boolean value of TRUE if it succeeds and of FALSE if it fails.

```
virtual BOOLEAN Insert(const CString&  
theString, int thePosition=-1);
```

Allows you to insert one string item at a time rather than the list of strings required by the preceding Insert method. Thus, it takes a single string and, by default, appends it to the end of any list of items already contained in the native list. Line numbers start at zero (0). A value of -1 for thePosition specifies that the insertion starts at the end of any other list. This method returns a Boolean value of TRUE if it succeeds and of FALSE if it fails.

```
virtual BOOLEAN Remove(int thePosition);
```

Takes an integer specifying a position in the native list and removes the item at that position. Line numbers start at zero (0). This method returns a Boolean value of TRUE if it succeeds and of FALSE if it fails.

```
virtual BOOLEAN Clear(void);
```

Removes (clears out) all the items from the native list and returns a Boolean value of TRUE if it succeeds and of FALSE if it fails.

Methods for Getting Items

```
virtual CStringList GetAllItems(void);
```

Returns a list of all the items in a native list.

```
virtual CString GetItem(int thePosition, int  
theNumberOfCharacters=ITEMLENGTH);
```

Returns the string item at the specified position in the native list. Line numbers start at zero (0). theNumberOfCharacters specifies the size of the buffer into which the item will be copied. The default buffer size is 255 characters.

```
virtual int GetNumItems(void);
```

Returns an integer indicating the number of items contained in the native list.

Protected Methods

Constructor, Destructor, and Initializer Methods

```
CNativeList(CSubview* theEnclosure, const  
CRect& theRegion, WIN_TYPE theControlType,  
const CStringList& theItems, const CString&  
theTitle, long theControlAttributes);
```


A constructor. `theEnclosure` is a pointer to the subview that will contain the native list. `theRegion` is a coordinate location, local to the enclosure, that is used to place the native list. `theControlType` specifies one of three possible types of native lists: `WC_LISTEDIT`, `WC_LISTBOX`, `WC_LISTBUTTON`. `theItems` is a list of the strings that the native list will contain. `theTitle` is the string that will serve as the title of the native list. Finally, `theControlAttributes` specifies the XVT Portability Toolkit attributes that determine the special characteristics and initial state of a control when it is created. The possible XVT Portability Toolkit control flags vary from control to control, and only a couple of them are generic to all controls. For information on the possible attributes a particular type of control can have, see the discussions of `create_control` and `CTL_FLAG` in the *XVT Programmer's Reference*.

```
CNativeList(const CNativeList& theNativeList);
```

A copy constructor. It duplicates the attributes of a native list. The items inside the native list are currently not being copied.

```
CNativeList&operator=(const CNativeList&  
theNativeList);
```

An assignment operator. It duplicates the attributes of a native list. The items inside the native list are currently not being copied.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);  
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);  
virtual Units* GetUnits(void) const;  
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Activate(void);  
virtual void Deactivate(void);  
virtual void DoActivate(void);  
virtual void DoCommand(Long theCommand, void* theData=NULL);  
virtual void DoDeactivate(void);
```

```
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
```

```

virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CNativeView

```

virtual void Disable(void);
virtual void DoHit(CONTROL_INFO theControlInfo) = NULL;
virtual void Enable(void);

```

```

WIN_TYPE GetXVTType(void) const;
WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview *thenclosure);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theId);
virtual void SetOrigin(const CPoint& diff);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void UpdateUnits(CUnits* theUnits);

```

Summary: CNativeList.h

```

#ifndef CNativeList_H
#define CNativeList_H

#include "PwrNames.h"
#include "PwrDef.h"

#include CNativeView_i

class CStringList;
class CString;

const int ITEMLENGTH = 255; // max length of the line item buffer in list

class CNativeList : public CNativeView -
{
public:
    // add & delete & clear
    virtual BOOLEAN Insert(const CStringList& theStringList,
        int thePosition=-1);

    virtual BOOLEAN Insert(const CString& theString,
        int thePosition=-1);

    virtual BOOLEAN Remove(int thePosition);

    virtual BOOLEAN Clear(void);

    // Get Items
    virtual CStringList GetAllItems(void);
    virtual CString GetItem(int thePosition,
        int theNumberOfCharacters=ITEMLENGTH);
    virtual int GetNumItems(void);

protected:

```

```
// constructor && destructor && init
CNativeList(CSubview* theEnclosure,
            const CRect& theRegion,
            WIN_TYPE theControlType,
            const CStringList& theItems,
            const CString& theTitle,
            long theControlAttributes);

CNativeList(const CNativeList& theNativeList);

CNativeList&operator=(const CNativeList& theNativeList);

};

#endif // CNativeList_H
```

CNativeSelectList



Description

CNativeSelectList is an abstract class that adds the functionality for selecting or deselecting one or more items in a native list. It includes functions for finding out whether a given item is selected or (conversely) finding the positions of selected items.

Heritage

Superclass: CNativeList

Subclasses: NListButton, NListBox

Usage

Override this base class with classes that are specific to each XVT Portability Toolkit native list control.

Environment

There are no environment settings for classes that inherit from CNativeView. The drawing properties of native views are system-defined.

Data Members

None.

Public Methods

Methods for Getting the Selection

```
virtual int GetNumSelectedItems(void);
```

Returns an integer indicating how many items are currently selected within the native select list.

```
virtual CStringList GetSelectedItems(void);
```

Returns a list of the string items that are currently selected within the native select list.

```
virtual CString GetFirstSelectedItem(int  
theNumberOfCharacters = ITEMLENGTH);
```

Returns the first selected string in the list.
theNumberOfCharacters specifies the size of the buffer into which the string will be copied. The default length is 255 characters.

```
virtual int GetSelectPosition(void);
```

Returns the line number of the first selected item in the list. Line numbers start at zero (0).

```
virtual BOOLEAN IsItSelected(int thePosition);
```

Given a line number, returns a Boolean value of TRUE if the item at that position is selected and FALSE if it is not.

Methods for Setting the Selection

```
virtual BOOLEAN SelectItem(int  
thePositionOfItem);
```

Given a line number (thePositionOfItem), selects the item at that position and returns a Boolean value of TRUE if it succeeds or FALSE if it fails.

```
virtual BOOLEAN DeselectItem(int  
thePositionOfItem);
```

Given a line number (thePositionOfItem), deselects the item at that position and returns a Boolean value of TRUE if it succeeds or FALSE if it fails.

```
virtual BOOLEAN SelectAll(void);
```

Selects all the items in the list and returns a Boolean value of TRUE if it succeeds or FALSE if it fails.

```
virtual BOOLEAN DeselectAll(void);
```

Deselects all the items in the list and returns a Boolean value of TRUE if it succeeds or FALSE if it fails.

Protected Methods

Constructor, Destructor, and Initializer Methods

```
CNativeSelectList(CSubview*theEnclosure, const  
CRect& theRegion, WIN_TYPE theControlType,  
const CStringList& theItems, const CString&  
theTitle, long theControlAttributes);
```

A constructor. `theEnclosure` is a pointer to the subview that will contain the native list. `theRegion` is a coordinate location, local to the enclosure, that is used to place the native list. `theControlType` specifies one of two possible types of native select lists: `WC_LISTBOX`, `WC_LISTBUTTON`. `theItems` is a list of the strings that the native select list will contain. `theTitle` is the string that will serve as the title of the native list. Finally, `theControlAttributes` specifies the XVT Portability Toolkit attributes that determine the special characteristics and initial state of a control when it is created. The possible XVT Portability Toolkit control flags vary from control to control, and only a couple of them are generic to all controls. For information on the possible attributes a particular type of control can have, see the discussions of `create_control` and `CTL_FLAG` in the *XVT Programmer's Reference*.

```
CNativeSelectList(const CNativeSelectList&
theNativeList);
```

A copy constructor. It duplicates the attributes of a native select list. The items inside the native select list are currently not being copied.

```
CNativeSelectList& operator=(const
CNativeSelectList& theNativeList);
```

An assignment operator. It duplicates the attributes of a native select list. The items inside the native select list are currently not being copied.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void DoActivate(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
```



```
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
```

```

virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CNativeView

```

virtual void Disable(void);
virtual void DoHit(CONTROL_INFO theControlInfo) = NULL;

```

```

virtual void Enable(void);
WIN_TYPE GetXVTType(void) const;
WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview *thenclosure);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theId);
virtual void SetOrigin(const CPoint& diff);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void UpdateUnits(CUnits* theUnits);

```

From CNative List

```

virtual BOOLEAN Clear(void);
virtual CStringList GetAllItems(void);
virtual CString GetItem(int thePosition,
    int theNumberOfCharacters=ITEMLENGTH);
virtual int GetNumItems(void);
virtual BOOLEAN Insert(const CStringList& theStringList, int thePosition=-1);
virtual BOOLEAN Insert(const CString& theString, int thePosition=-1);
virtual BOOLEAN Remove(int thePosition);

```

Summary: CNativeSelectList.h

```

#ifndef CNativeSelectList_H
#define CNativeSelectList_H

#include "PwrNames.h"
#include "PwrDef.h"

#include CNativeList_i

class CStringList;
class CString;

class CNativeSelectList : public CNativeList
{
public:

```

```
// getting the selection
virtual int GetNumSelectedItems(void);
virtual CStringList GetSelectedItems(void);
virtual CString GetFirstSelectedItem(int theNumberOfCharacters =
    ITEMLENGTH);
virtual int GetSelectPosition(void);
virtual BOOLEAN IsItSelected(int thePosition);

// setting the selection
virtual BOOLEAN SelectItem(int thePositionOfItem);
virtual BOOLEAN DeselectItem(int thePositionOfItem);
virtual BOOLEAN SelectAll(void);
virtual BOOLEAN DeselectAll(void);

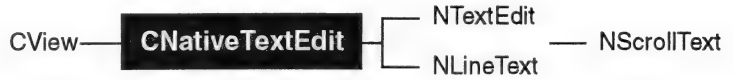
protected:

// constructor && destructor && init
CNativeSelectList(CSubview* theEnclosure,
    const CRect& theRegion,
    WIN_TYPE theControlType,
    const CStringList& theItems,
    const CString& theTitle,
    long theControlAttributes);

CNativeSelectList(const CNativeSelectList& theNativeList);
CNativeSelectList& operator=(const CNativeSelectList& theNativeList);
};

#endif // CNativeSelectList_H
```

CNativeTextEdit



Description

CNativeTextEdit is the abstract text editing class that provides functionality for native text edit boxes. The CNativeTextEdit branch of CView represents text edit objects, allowing the user to display text inside views, type in text, and perform the usual cut, paste, and copy operations. All CNativeTextEdit objects inherit the properties of CView, such as sizing, dragging, and so on. As an abstract class, CNativeTextEdit represents a rectangular text edit area on the screen and provides basic text manipulation methods. There is no concept of lines, paragraphs, or scrolling at this level. These text properties are embodied in the various classes that are derived from CNativeTextEdit, classes that represent the different types of text editing systems available in XVT-Power++.

Heritage

Superclass: CView

Subclasses: NTextEdit, NLineText

Usage

This is an abstract class. You must derive your own classes from it or use such classes as NLineText or NTextEdit.

Environment

The text is drawn in the foreground color; the border is drawn with the *pen* color. The interior of the text box is drawn in the background color.

Public Data Members

None.

Protected Data Members

TXEDIT itsTextEdit; the native view

Private Data Members

BOOLEAN	itAutoSelects;	whether there is auto selection
static WINDOW	itsUpdateWindow;	the window needs updating

Public Methods

Destructor and Initializer Methods

```
virtual ~CNativeTextEdit(void);
```

The destructor, which cleans up the text edit box.

```
BOOLEAN INativeTextEdit(unsigned theAttributes = TX_BORDER,
                        UNITS theRightMargin    = 1000,
                        int theCharacterLimit    = 1000,
                        const CString theInitialText = NULLString,
                        BOOLEAN isAutoSelected    = FALSE,
                        BOOLEAN isVisible         = TRUE,
                        GLUETYPE theGlue         = NULLSTICKY);
```

The initializer. It takes an unsigned value pertaining to the XVT Portability Toolkit attributes that the text edit system can have. These attributes can be ORed and passed in together, as described in the text editing chapter of the *XVT Programmer's Guide*. Possible XVT Portability Toolkit values for theAttributes are shown in the following table:

TX_AUTOHSCROLL	Enables horizontal automatic scrolling when the user drags the mouse outside of the view rectangle
TX_AUTOVSCROLL	Enables vertical automatic scrolling
TX_BORDER	Draws a rectangular border around the view rectangle.
TX_ENABLECLEAR	Leaves the clear item in the Edit menu always enabled.
TX_NOCOPY	Does not enable the Copy command on the Edit menu
TX_NOCUT	Does not enable the Cut command on the edit menu
TX_NOMENU	Prevents the text editing system from changing the menu of the window containing the text editing object. This attribute is useful if there is no Edit menu.

TX_NOPASTE	Does not enable the Paste command on the Edit menu
TX_ONEPAR	Limits editing to one paragraph; ignores carriage returns
TX_OVERTYPE	Enables "overtyping" mode that allows users to replace existing characters when typing instead of inserting characters in front of existing text.
TX_READONLY	Does not allow changes to the text except from the application program
TX_WRAP	Enables word wrap to the margin

The next parameter, `theRightMargin` is a pixel measurement of the right margin, starting from the left side and beginning at zero (0). In addition, there is a parameter that sets the default character limit at 1,000 characters. `theInitialText` is a string containing any text you may wish to initialize with the text edit system. If the `isAutoSelected` parameter is set to `TRUE`, all of the text in the text edit box becomes selected when the user clicks inside the box. Finally, this initializer, like the initializers of all `CView` classes, takes a visibility state and a glue type.

Text Utilities

virtual void SetText(const CString& theText);

Given a text string, `SetText` sets the text for the text edit box. Any text already inside the box is replaced. This method sets the text to be exactly what is passed in.

virtual BOOLEAN Append(const CString& theText);

Given a text string, `Append` appends this string to whatever text is already inside a text edit object. This method returns a Boolean value of `TRUE` if the append operation succeeds.

virtual CString GetText(void) const;

Returns a string containing all of the text in the edit box.

virtual T_CNUM GetNCharInText(void) const;

Returns the number of characters in a text edit box.

Selection Utilities

virtual void SelectText(void);

Selects all of the text inside a text edit box.

virtual CString GetSelectedText(void) const;

This method is analogous to `GetText`, except that it applies to selected text. That is, it returns a string containing all selected text in the edit box.

```
virtual T_CNUM GetNCharInSelection(void) const;
```

Returns the number of characters in the selected text within a text edit box.

NonText Utilities

```
virtual BOOLEAN Clear(void);
```

Clears the contents of a text box.

```
BOOLEAN IsEmpty(void) const;
```

Returns a Boolean value indicating whether a text box contains any text.

```
virtual void SetAttribute(unsigned  
theAttribute, BOOLEAN isSet=TRUE);
```

Sets an attribute for a text box. You specify the attribute and give it a Boolean value indicating whether to turn this attribute on or off. See `INativeTextEdit` for further information about the XVT Portability Toolkit attributes. Also, see the text editing chapter of the *XVT Programmer's Guide*.

```
virtual unsigned long GetAttributes(void)  
const;
```

Returns the attributes of a text editing box.

```
virtual void SetLimit(int theCharacterLimit);
```

Sets the limit on the number of characters a text box can contain. If you set a new limit and the text box already contains more characters than the newly set limit, the text is not truncated. For example, if you set a new limit of 400 characters for a text box that already contains 600 characters, then the 600 characters remain. However, no more text can be entered into the box.

```
virtual int GetLimit(void) const;
```

Returns the maximum number of characters a text box can contain.

```
virtual void SetMargin(UNITS theRightMargin);
```

Sets the right margin of a text edit box. `theRightMargin` is a pixel measurement of the right margin, starting from the left side and beginning at zero (0). If the text edit object has a

horizontal scrollbar, this method resets the scrollbar's range according to the new margin.

virtual UNITS GetMargin(void) const;

Returns a number indicating the pixel measurement of a text box's right margin. Numbering starts at zero (0) at the left margin.

virtual void Reset(void);

Redraws the entire text box and scrolls its contents to the top of the box. It also sets the cursor at the top-left corner of the text box.

virtual void Suspend(void);

Suspends the redrawing of a text box's contents so that multiple text insertions can be done in rapid succession without taking the time to redraw the box after each insertion. (See Resume.)

virtual void Resume(void);

Resumes the redrawing of a text edit box's contents after redrawing has been suspended. It is usually a good idea to "wrap" the insertion of text into a text edit box by calls to Suspend and Resume. (See Suspend.)

Key Validation Handling

virtual int Validate(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);

A method that is called when a text box receives a keyboard event. *theKey* is the ASCII value for the character key that was pressed; *isShiftKey* and *isControlKey* indicate whether the SHIFT or CONTROL key was pressed in conjunction with the character key.

Validate responds to a keyboard event in one of three ways:

1. Validate accepts the keyboard event and returns *theKey* as the ASCII value of the character to be placed in the text edit system.
2. Validate rejects the keyboard event by returning NULL.
3. Validate seeks to modify the keyboard event by returning the ASCII value of the character it wishes to pass on to the text edit system.

To see how Validate works, think of implementing a password text box. When the user enters a login name, Validate would

behave as in case 1 by permitting the characters being typed to appear on the screen. However, when the user enters a password, `Validate` may choose to modify the input by returning an asterisk (*) for each event received. In addition, `Validate` may behave as in case 2, ignoring all tabs by returning `NULL`.

Override this method to implement the kind of text validation described here. By default, this method always returns the exact key that it receives.

Inherited Utilities

```
virtual void Key(int theKey, BOOLEAN  
isShiftKey, BOOLEAN isControlKey);
```

Sends a keyboard event to a text edit box. `theKey` is the ASCII value for the character key that was pressed; `IsShiftKey` and `IsControlKey` indicate whether the SHIFT or CONTROL key was pressed in conjunction with the character key. This method calls `Validate` to allow the user to validate the key before actually passing it on to the text edit system.

```
virtual void SetOrigin(const CPoint& theDelta);
```

Takes the current origin of the text edit box and adds `theDeltaPoint` to it. Suppose, for example, that a text box has an origin of 10,20 and you want to shift the origin by three pixels to the right. You set the origin by giving it a delta point of 3,0, which shifts the view three pixels horizontally.

```
virtual void SetFont(const FONT& theNewFont,  
BOOLEAN isUpdate = FALSE);
```

Takes a font and sets the text box's environment to have that font. The `isUpdate` parameter indicates whether this `SetFont` message is simply an update message or whether it is a "real" `SetFont` message meaning that the text box should set its own font to the new font.

```
virtual void Size(const CRect& theNewSize);
```

Resets the region of the text box. Possibly the region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, `theNewSize`, like the region that is passed in on creation of a text box, is relative to the enclosure.

```
virtual void Show(void);
```

Makes the text edit box and its contents visible.

```
virtual void Hide(void);
```

Makes the text edit box and its contents invisible.

```
virtual void Activate(void);
```

Sets the state of the text box to active so that it can receive events.

```
virtual void Draw(const CRect&  
theClippingRegion);
```

Takes care of any drawing that the text box must do. theClippingRegion is the part of the text box that needs to be drawn. If it is set to NULL, the entire text box is drawn. theClippingRegion is in global, window-relative coordinates.

```
virtual void SetGlue(GLUETYPE theGlue);
```

Sets the glue type of the glue object associated with the text box. By default, a glue object has a type of NULLSTICKY upon creation. See the table under “Usage” in the section on CGlue for a listing of valid glue types.

```
virtual void Disable(void);
```

Disables a text box so that it cannot receive events.

```
virtual void SetEnvironment(const CEnvironment&  
theNewEnvironment, BOOLEAN isUpdate =  
FALSE);
```

Sets the environment for a text box, using theNewEnvironment that is passed to it. By default, text boxes share their enclosure’s environment. However, as soon as you use SetEnvironment to give a text box an environment of its own, the text box uses that environment instead of the shared environment.

Mouse Events

CNativeTextEdit overrides two mouse methods to implement its selection behavior. Mouse events are passed directly to the text box at the location where the event occurs. Each of the following methods has a theButton argument that specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right). By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button.

```
virtual void MouseDouble(CPoint theLocation,  
short theButton = 0,  
BOOLEAN isShiftKey = FALSE,  
BOOLEAN isControlKey = FALSE);
```

When the user double clicks a mouse button over a text box containing local coordinate `theLocation`, the text box receives and handles this event.

```
virtual void MouseDown(CPoint theLocation,
                      short theButton      = 0,
                      BOOLEAN isShiftKey   = FALSE,
                      BOOLEAN isControlKey = FALSE);
```

When the user presses down a mouse button over a text box containing local coordinate `theLocation`, the text box receives and handles this event.

```
static void ProcessEvents(WINDOW win =
                        NULL_WIN, EVENT* ep = NULL);
```

A method called by XVT-Power++'s switchboard to pass events to the XVT Portability Toolkit text system. If you override the switchboard, call this method and pass in the same parameters that are passed to the event handler.

```
static void SetUpdateWindow(const CWindow&
                          theWindow);
```

A method that can be called to indicate that a certain window's text editing boxes need to be updated.

Protected Methods

Constructor Methods

```
CNativeTextEdit(CSubview *theEnclosure,
                const CRect& theRegion,
                unsigned theAttributes= TX_BORDER,
                UNITS theRightMargin = 1000,
                int theCharacterLimit = 1000);
```

A constructor. `theEnclosure` is a pointer to the subview that will contain the text box. `theRegion` is a coordinate location, local to the enclosure, that is used to place the text box. If the text box has a border, which is the default, then the border rectangle is drawn around the `CRect`. This `CRect` object is inset by 4 pixels inside the border. The bottom of the `CRect` may be inset even more to ensure that an integral number of lines will fit into the text box.

In addition, this constructor takes an unsigned value pertaining to the XVT Portability Toolkit attributes that the text edit system can have. These attributes can be ORed and passed in together, as described in the text editing chapter of the *XVT Programmer's Guide*. See `INativeTextEdit` for further information about the XVT Portability Toolkit attributes.

Finally, this constructor takes a right margin for the text box, which is measured in logical units starting with zero (0) at the left side of the text box, and a limit on the number of characters that can fit into the text box.

```
CNativeTextEdit(const CNativeTextEdit&
theTextEdit);
```

A copy constructor that duplicates the attributes of a text box, as well as the text inside it.

```
CNativeTextEdit& operator=(const
CNativeTextEdit& theTextEdit);
```

An assignment operator that duplicates the attributes of a text box, as well as the text inside it.

Internal Text Manipulation

The sole purpose of the methods in this section is to be used as utilities by classes that inherit from CNativeTextEdit.

```
CString GetTextInternal(T_PNUM theFirstParagraph,
                        T_PNUM theLastParagraph,
                        T_LNUM theFirstLine,
                        T_LNUM theLastLine,
                        T_CNUM theFirstChar,
                        T_CNUM theLastChar,
                        CString theTextBuffer) const;
```

Returns all of the text contained in a text box. It is very similar to the copy constructor GetText, but it performs the actual calculations within the text editing system. GetTextInternal takes the numbers of the first and last paragraphs, the numbers of the first and last lines, and the numbers of the first and last characters. Then it appends the text mapped out by these parameters to any text already contained in the given string buffer.

```
T_CNUM GetNCharInternal(T_PNUM theFirstParagraph,
                        T_PNUM theLastParagraph,
                        T_LNUM theFirstLine,
                        T_LNUM theLastLine,
                        T_CNUM theFirstChar,
                        T_CNUM theLastChar) const;
```

Returns the number of characters of all the text contained in a text box. GetNCharInternal takes the numbers of the first and last paragraphs, the numbers of the first and last lines, and the numbers of the first and last characters. Then it counts all the characters in the chunk of text mapped out by these parameters.

```
void GetFullPar(T_PNUM theParagraph, CString  
theText) const;
```

Given a paragraph number, returns the full paragraph of text, appending it to any text already contained in the given string buffer.

```
void GetLineInternal(T_PNUM theParagraph,  
T_LNUM theLine, CString theText) const;
```

Returns one line of text. This method takes the paragraph number and the line number and then appends the line to any text already contained in the given string buffer.

```
void Truncate(CString& theText);
```

An internal method that truncates the text to fit the character limit of the CNativeTextEdit object.

```
void ResetColors(void);
```

An internal method that is called to update the colors of the text after a call to SetEnvironment.

```
virtual void UpdateUnits(CUnits* theUnits);
```

Updates the CUnits object indicated by theUnits, which is the CUnits object owned by the CNativeTextEdit object, that is, an object inheriting from CNativeTextEdit.

See Also: CUnits

Private Methods

```
void GetPartLine(T_CNUM theFirstChar, T_CNUM theLastChar,  
T_PNUM theParagraph,  
T_LNUM theLineNumber,  
CString theText) const;
```

Returns part of a line. This method takes the numbers of the starting and ending characters in the line, the paragraph number, the line number, and a string buffer. It appends the partial line mapped out by the first four parameters to any text contained in the string buffer.

```
void GetPartPar(T_CNUM theStartChar, T_CNUM theEndChar,  
T_PNUM theParagraph,  
T_LNUM theStartLine, T_LNUM theEndLine,  
CString theText) const;
```

Returns part of a paragraph. It has parameters for the starting character and ending character, the paragraph number, the starting line and ending line, and a string buffer. The partial

paragraph mapped out by the first five parameters is appended to any text already contained in the string buffer.

```
T_CNUM GetNCharInPartPar(T_CNUM theStartChar,
                        T_CNUM theEndChar,
                        T_PNUM theParagraph,
                        T_LNUM theStartLine,
                        T_LNUM theEndLine) const;
```

Returns the total number of characters in part of a paragraph. It takes the starting character and the ending character in the partial paragraph, the paragraph number, and the numbers of the starting line and ending line.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Deactivate(void);
virtual void DoActivate(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
```

```
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
```



```

virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetId(int theID);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

Summary: CNativeTextEdit.h

```

#ifndef CNativeTextEdit_H
#define CNativeTextEdit_H

#include "xvt.h"
#include "CView.h"

class CNativeTextEdit : public CView
{
public:
    // construc & destruct & init
    virtual ~CNativeTextEdit(void);
    BOOLEAN INativeTextEdit(unsigned theAttributes    = TX_BORDER,
        UNITS theRightMargin    = 1000,
        int theCharactedLimit    = 1000,
        const CString theInitialText    = NULLString,
        BOOLEAN isAutoSelected    = FALSE,
        BOOLEAN isVisible    = TRUE,
        GLUETYPE theGlue    = NULLSTICKY);

    // Text utilities:
    virtual void SetText(const CString& theText);
    virtual BOOLEAN Append(const CString& theText);

    virtual CString GetText(CString theText) const;
    virtual CString GetTextAlloc(void) const;

    virtual T_CNUM GetNCharInText(void) const;

```

```

// Selection utilities:
virtual void SelectText(void);

virtual CString GetSelectedText(CString theTextBuffer) const;
virtual T_CNUN GetNCharInSelection(void) const;

// Other:
virtual BOOLEAN Clear(void);
virtual BOOLEAN IsEmpty(void) const;

virtual void SetAttribute(unsigned theAttribute,
    BOOLEAN isSet=TRUE);
virtual unsigned long GetAttributes(void) const;

virtual void SetLimit(int theCharacterLimit);
virtual int GetLimit(void) const;

virtual void SetMargin(UNITS theRightMargin);
virtual UNITS GetMargin(void) const;

virtual void Reset(void);

virtual void Suspend(void);
virtual void Resume(void);

// Key validation handling:
virtual int Validate(int theKey, BOOLEAN isShiftKey,
    BOOLEAN isControlKey);

// Inherited utilities:
virtual void Key(int theKey, BOOLEAN isShiftKey,
    BOOLEAN isControlKey);
virtual void SetOrigin(const CPoint& theDelta);
virtual void SetFont(const FONT& theNewFont,
    BOOLEAN isUpdate = FALSE);
virtual void Size(const RECT& theNewSize);
virtual void Show(void);
virtual void Hide(void);
virtual void Activate(void);
virtual void Draw(const RECT& theClipping);
virtual void SetGlue(GLUETYPE theGlue);

virtual void Enable(void);
virtual void Disable(void);

virtual void SetEnvironment(const CEnvironment& anEnv,
    BOOLEAN isUpdate = FALSE);

// Mouse and Key events:
virtual void MouseDouble(CPoint theLocation, short theButton=0,
    BOOLEAN isShiftKey = FALSE,
    BOOLEAN isControlKey = FALSE);
virtual void MouseDown(CPoint theLocation, short theButton=0,
    BOOLEAN isShiftKey = FALSE,
    BOOLEAN isControlKey = FALSE);

// Call the following to pass events into the text system:
static void ProcessEvents(WINDOW win = NULL_WIN, EVENT* ep = NULL);
static void SetUpdateWindow(const CWindow& theWindow);

```

protected:

```
    TXEDIT itsEditText; // the XVT Portability Toolkit control
```

```

// construc & destruct & init
CNativeTextEdit(CSubview *theEnclosure, const CRect& theRegion,
    unsigned theAttributes = TX_BORDER,
    UNITS theRightMargin = 1000,
    int theCharacterLimit = 1000);
CNativeTextEdit(const CNativeTextEdit& theTextEdit);
CNativeTextEdit& operator=(const CNativeTextEdit& theTextEdit);

// Internal text manipulation:

CString GetTextInternal(T_PNUM theFirstParagraph,
    T_PNUM theLastParagraph,
    T_LNUM theFirstLine,
    T_LNUM theLastLine,
    T_CNUM theFirstChar,
    T_CNUM theLastChar,
    CString theTextBuffer) const;

T_CNUM GetNCharInternal(T_PNUM theFirstParagraph,
    T_PNUM theLastParagraph,
    T_LNUM theFirstLine,
    T_LNUM theLastLine,
    T_CNUM theFirstChar,
    T_CNUM theLastChar) const;

void GetFullPar(T_PNUM theParagraph, CString theText) const;

void GetLineInternal(T_PNUM theParagraph, T_LNUM theLine,
    CString theText) const;

void Truncate(CString& theText);
void ResetColors(void);
virtual void UpdateUnits(CUnits* theUnits);

private:
    BOOLEAN itAutoSelects; // wants auto selection
    static WINDOW itsUpdateWindow; // the window needs updating

    void GetPartLine(T_CNUM theFirstChar, T_CNUM theLastChar,
        T_PNUM theParagraph, T_LNUM theLineNumber,
        CString theText) const;

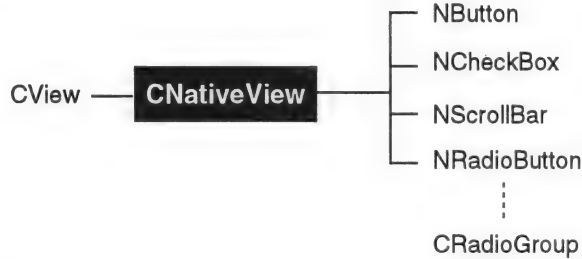
    void GetPartPar(T_CNUM theStartChar, T_CNUM theEndChar,
        T_PNUM theParagraph,
        T_LNUM theStartLine, T_LNUM theEndLine,
        CString theText) const;

    T_CNUM GetNCharInPartPar(T_CNUM theStartChar, T_CNUM theEndChar,
        T_PNUM theParagraph,
        T_LNUM theStartLine, T_LNUM theEndLine) const;
};

#endif CNativeTextEdit_H

```

CNativeView



Description

CNativeView is an abstract class from which several view classes are derived. Native views are the means of communication between the application and the user who is operating the mouse. Inheriting from CNativeView are objects such as buttons, scrollbars, check boxes, and radio buttons, which have the look and feel of the native window manager.

Heritage

Superclass: CView

Subclasses: NButton, NCheckBox, NScrollBar, NRadiobutton

Usage

This is a base class that must be overridden by classes specific to each native view.

Environment

There are no environment settings for classes that inherit from CNativeView. The drawing properties of native views are system-defined.

Public Data Members

None.

Protected Data Members

long	itsXVTAttributes;	union of XVT Portability Toolkit CTL_* flags
------	-------------------	--

Private Data Members

static CNativeView *selectedControl;		implement dragging in WSMTF (Motif only)
static CList	movableControls;	implement dragging in WSMTF (Motif only)
WIN_TYPE	itsType;	the native view's type
BOOLEAN	itIsEnabled;	whether the native view can receive events
WINDOW	itsNativeView;	the window handler for the native control

Friends

friend class	CControlWireFrame;	the private overridden version of C WireFrame used by all native views
--------------	--------------------	--

Public Methods

Destructor and Initializer Methods

```
virtual ~CNativeView(void);
```

The destructor, which cleans up after native views.

```
BOOLEAN INativeView(const CString theTitle= NULLString,
                     BOOLEAN isEnabled   = TRUE,
                     BOOLEAN isVisible    = TRUE,
                     GLUETYPE theGlue    = NULLSTICKY);
```

The initializer. It takes a title, a Boolean value indicating whether it is enabled and thus can receive events, a visibility status, and a glue type.

Communication

```
virtual void DoHit(CONTROL_INFO theControlInfo)
= NULL;
```

The primary event-receiving method for native views. It takes a parameter of type CONTROL_INFO, which is defined by the XVT Portability Toolkit in the *XVT Programmer's Guide* as follows:

```

typedef struct s_ctlinfo {
    WIN_TYPE type; /* WC_ */
    WINDOW win; /* WINDOW of control */
    union {
        struct s_pushbutton {
            int reserved; /* Reserved (unused). */
        } pushbutton;

        struct s_radiobutton {
            int reserved; /* Reserved (unused). */
        } radiobutton;

        struct s_checkbox {
            int reserved; /* Reserved (unused). */
        } checkbox;

        struct s_scroll {
            SCROLL_INFO what; /* activity site */
            short_pos; /* thumb position */
        } scroll;

        struct s_edit {
            BOOLEAN focus_change; /* is event a focus change? */
            BOOLEAN active; /* if so: gaining focus? */
        } edit;

        struct s_statictext {
            int reserved; /* Reserved (unused). */
        } statictext;

        struct s_lbox {
            BOOLEAN dbl_click; /* double click? */
        } lbox;

        struct s_listbutton {
            int reserved; /* Reserved (unused). */
        } listbutton;

        struct s_listedit {
            BOOLEAN focus_change; /* Did the edit field part change
                                   focus? */
            BOOLEAN active; /* if so: gaining focus? */
        } listedit;

        struct s_groupbox {
            int reserved; /* Reserved (unused). */
        } groupbox;

        struct s_icon {
            int reserved; /* Reserved (unused). */
        } icon;
    } v;
} CONTROL_INFO, *CONTROL_INFO_PTR;

```

This method must be overridden by derived classes.

Inherited Utilities

```
virtual void Size(const CRect& theNewSize);
```

Resets the region of the native view. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of

the region, theNewSize, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated.

virtual void SetOrigin(const CPoint& diff);

Takes the current origin of the native view and adds theDeltaPoint to it. Suppose, for example, that a native view has an origin of 10,20 and you want to shift the origin by three pixels to the right. You set the origin by giving it a delta point of 3,0, which shifts the native view three pixels horizontally.

virtual void Hide(void);

Makes the native view invisible.

virtual void Show(void);

Makes the native view visible.

virtual void SetId(int theId);

Sets the ID number of the native view.

**virtual void SetEnclosure(CSubview
*theEnclosure);**

Sets the enclosure for the native view to a different enclosure. Note that if a native view belongs to a certain enclosure and then it is placed inside of a different enclosure, its location relative to the window is also going to change.

**virtual void SetTitle(const CString&
theNewTitle);**

Gives the native view the title designated by theNewTitle.

virtual void Enable(void);

Enables the native view so that it can receive events.

virtual void Disable(void);

Disables the view so that it cannot receive events.

Inherited Sizing and Dragging Methods

In the case of native views (controls), setting the moving and sizing to TRUE in effect disables a view because that view no longer acts as it usually does—as a button, for example. Instead, it simply acts as an object that is being moved and sized interactively by the user.

virtual void SetSizing(BOOLEAN isSizable);

Takes a Boolean value that sets the sizing of a native view to TRUE so that the view can be sized or to FALSE so that the view cannot be sized.

```
virtual void SetDragging(BOOLEAN isDraggable);
```

Takes a Boolean value that sets the dragging of a native view to TRUE so that the view can be dragged with the mouse or to FALSE so that it cannot be dragged.

Protected Methods

Constructor and Initializer Methods

```
CNativeView(CSubview* theEnclosure,  
            const CRect& theRegion,  
            WIN_TYPE theControlType,  
            long theControlAttributes= NULL;  
            const CString& theTitle = NULLString);
```

The constructor. It takes a pointer to an enclosure and a region that is a CRect. The enclosure is the view that contains the native view; all native views must be placed within an enclosure. theRegion is a coordinate location that is local to the enclosure; this region indicates where the new native view is to be placed. This constructor also takes a control type, which is an XVT Portability Toolkit window type. Possible XVT Portability Toolkit values for theControlType are shown in the following table.

WC_PUSHBUTTON	button control
WC_EDIT	edit field
WC_LBOX	list box
WC_ICON	icon
WC_VSCROLL	vertical scrollbar
WC_HSCROLL	horizontal scrollbar
WC_TEXT	static text
WC_LISTEDIT	list-edit combo control
WC_LISTBUTTON	list-button combo control
WC_CHECKBOX	check box
WC_RADIOBUTTON	radio button
WC_GROUPBOX	groupbox
WO_TE	XVT Portability Toolkit text edit object

theControlAttributes takes a value from a set of XVT Portability Toolkit-provided attributes that you can give to native views. You can OR together the appropriate control flag constants into an attribute value. The possible control flags for theControlAttributes are shown in the following table. In addition to the ones shown here, the XVT Portability Toolkit supplies other, platform-specific flags.

CTL_FLAG_DISABLED	initially disabled control
CTL_FLAG_CHECKED	the “checked” state for check boxes and radio buttons only
CTL_FLAG_DEFAULT	default border style for push buttons in a dialog
CTL_FLAG_INVISIBLE	initially invisible
CTL_FLAG_GROUP	the first or last element in a radio button grouping, for keyboard navigation purposes
CTL_FLAG_READONLY	a readable but not selectable list box
CTL_FLAG_MULTIPLE	multiple selections in a list box
CTL_FLAG_NATIVE_JUST	native text justification
CTL_FLAG_LEFT_JUST	left text justification
CTL_FLAG_CENTER_JUST	center text justification
CTL_FLAG_RIGHT_JUST	right text justification

Finally, this constructor takes a character string as a title for the native view.

```
CNativeView(WINDOWtheCreatedControl, CSubview*  
theEnclosure, const CRect& theRegion,  
WIN_TYPE theControlType);
```

A constructor that is to be used with derived classes that create a native control themselves. theCreatedControl is a handle to the control that is created by a derived class. theEnclosure is a pointer to the subview that will contain the control. theRegion is a coordinate location, local to the enclosure, that is used to place the control. theControlType is the type of control that is created, that is, whether it is a scrollbar, a radio button, a check box, and so on.

```
CNativeView(const CNativeView& theControl);
```

A copy constructor. It duplicates the attributes of a native view.

```
CNativeView& operator=(const CNativeView&  
theControl);
```

An assignment operator. It duplicates the attributes of a native view.

WIN_TYPE GetXVTType(void) const;

Returns the XVT Portability Toolkit window type of the native view. For a listing of the possible values, see the table under the constructor method.

virtual void UpdateUnits(CUnits* theUnits);

Updates the CUnits object indicated by theUnits, which is the CUnits object owned by the CNativeView object.

See Also: CUnits.

Private Methods

void Close(void);

An internal method that is called upon the destruction of a native view.

void CreateControl(void);

An internal method that is called upon the construction of a native view.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

virtual void ChangeFont(FONT theFont, FONT_PART theChange);

virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);

virtual CUnits* GetUnits(void) const;

virtual void SetUnits (CUnits* theCoordinateUnits);

From CView

virtual void Activate(void);

virtual void Deactivate(void);

virtual void DoActivate(void);

virtual void DoCommand(long theCommand, void* theData=NULL);

virtual void DoDeactivate(void);

virtual void DoDisable(void);

virtual void DoDraw(void);

```
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
```

```

virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

Overrides

DoHit. In addition, you can override Size and Relocate methods to handle control-specific utilities, but the inherited function must still be called to physically move/resize the native view.

Summary: CNativeView.h

```

#ifndef CNativeView_H
#define CNativeView_H

#ifdef WSMTF
// In Motif, a special event hook is set up for moving/sizing.
#include "EventHook.h"
#endif WSMTF

#include "PwrDef.h"
#include CView_i
#include CWireFrame_i

class CWindow;
class CSubview;

class CNativeView : public CView
{
public:
    virtual ~CNativeView(void);

    BOOLEAN INativeView(const CString theTitle    = NULLString,
                        BOOLEAN isEnabled         = TRUE,
                        BOOLEAN isVisible          = TRUE,
                        GLUTYPE theGlue           = NULLSTICKY);

    // Communication:
    virtual void DoHit(CONTROL_INFO theControlInfo) = NULL;

    // Inherited utility:
    virtual void Size(const CRect& theNewSize);
    virtual void SetOrigin(const CPoint& diff);

    virtual void Hide(void);
    virtual void Show(void);

    virtual void SetId(int theId);
    virtual void SetEnclosure(CSubview *theEnclosure);

    virtual void SetTitle(const CString& theNewTitle);

    virtual void Enable(void);
    virtual void Disable(void);

    // Sizing and dragging (inherited):
    virtual void SetSizing(BOOLEAN isSizable);
    virtual void SetDragging(BOOLEAN isDraggable);

protected:
    long itsXVTAttributes; //union of XVT CTL_* flags

    // Constructors
    CNativeView(CSubview* theEnclosure,
                const CRect& theRegion,
                WIN_TYPE theControlType,
                long theControlAttributes = NULL,
                const CString& theTitle = NULLString);

    CNativeView(const CNativeView& theControl);
    CNativeView& operator=(const CNativeView& theControl);

    WINDOW GetXVTWindow(void) const;
    WIN_TYPE GetXVTType(void) const;

```

```

        virtual void UpdateUnits(CUnits* theUnits);
private:
        friend class CControlWireFrame;

#ifdef WSMTF
        friend BOOLEAN event_hook(XEvent* xevent);
        friend class CApplication;
        static CNativeView* selectedControl; // implement dragging in WSMTF
        static CList movableControls; // implement dragging in WSMTF
#endif WSMTF

        WIN_TYPE itsType; // the Type of control it is
        BOOLEAN itIsEnabled; // cant it receive events
        WINDOW itsNativeView; // the XVT control

        void Close(void);
        void CreateControl(void);
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#ifdef WSMTF
class CControlWireFrame : public CWireFrame
{
        // This is a CNativeView private Frame class used for sizing

        friend class CNativeView;
        friend BOOLEAN event_hook(XEvent* xevent);

private:
        CControlWireFrame(CNativeView* theOwner);

        Window itsXWindow;
        Window itsXRoot;

        virtual void MouseDown(CPoint theLoc, short theButton = 0,
                                BOOLEAN isShift=FALSE,
                                BOOLEAN isControl=FALSE);

        virtual void MouseUp(CPoint theLoc, short theButton = 0,
                               BOOLEAN isShift=FALSE,
                               BOOLEAN isControl=FALSE);
};

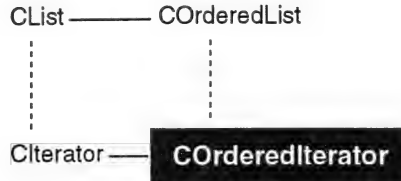
#endif WSMTF

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
inline WINDOW CNativeView::GetXVTWindow(void) const
{
        return (itsNativeView);
}

#endif CNativeView_H

```

COrderedIterator



Description

COrderedIterator is a class that iterates over COrderedList. COrderedList is similar to CList, except that the items in the list have an order. A position is associated with each element of the list. COrderedIterator iterates through the elements of the list in ascending position order. For more information, see COrderedList.

Heritage

Superclass: CIterator

Usage

Since templates are not yet supported by every C++ compiler, objects are inserted via void pointers. Thus COrderedList stores objects by their address. It does not store "copies" of objects. Because of the use of void*, you must use casting in at least two places when using COrderedList under the current non-template implementation. Here is an example of use:

```

CList aList();
Foo* aFoo = new Foo(1,2,3);
aList.Insert((void*) aFoo); // Cast #1
Foo* fooItem;
CIterator doTo(aList);
while (fooItem = (aFoo*) doTo.Next()) // Cast #2
  fooItem->Goo();
  
```

Public Data Members

None.

Protected Data Members

None.

Private Data Members

`CLink* itsCurrentPosLink;` keeps track of the position of each item on the list

Public Methods

`COrderedIterator(const COrderedList& theList);`

A constructor, which takes a reference to a list.

`COrderedIterator(const COrderedList* theList);`

A constructor, which takes a pointer to a list.

`int GetCurrentPosition(void) const;`

At any point during an iteration over an ordered list, you can use this method to find out the position number of the current item. Thus, this method provides a way to find out how the iteration is progressing.

Iterating Methods

The following methods are inherited from `CIterator` and overridden.

`virtual void* Next(void);`

Returns the next item on the list each time it is called until it reaches the end of the list, in which case it returns `NULL`. You can iterate by starting a loop and continuing the loop as long as `NEXT` returns non-`NULL` values. Here is an example of use:

```
CList aList();
Foo* aFoo = new Foo(1,2,3);
aList.Insert((void*) aFoo); // Cast #1
Foo* fooItem;
CIterator doTo(aList);
while (fooItem = (aFoo*) doTo.Next()) // Cast #2
    fooItem->Goo();
```

`virtual void* Previous(void);`

Returns the preceding item on the list each time it is called until it reaches the top of the list, in which case it returns `NULL`.

`virtual void* First(void);`

Returns the first item on the list and sets the iterator to the first element.

`virtual void* Last(void);`

Returns the last item on the list and sets the iterator to the last element. If you were to call Next after calling this method, the result would be NULL.

Summary: COrderedList.h

Note: COrderedIterator and COrderedList share the same header file. This file is shown in the discussions of each of these classes.

```
#ifndef COrderedList_H
#define COrderedList_H
#include "PwrNames.h"
#include "CLists.h"

class COrderedList : private CList
{
public:
    // Class Utility:
    COrderedList(void);
    COrderedList(const COrderedList& theOtherList);
    COrderedList& operator=(const COrderedList& theOtherList);

    // List Operations:
    BOOLEAN Insert(void* theItem, int thePosition);
    void* Replace(void* theItem, int thePosition);
    BOOLEAN Remove(void* theItem);
    void* Remove(int thePosition);

    void* Find(int thePosition) const;
    void Renumber(int theBasePosition, int theIncrement);

    // Privately inherited methods which are made available:
    CList::Clear();
    CList::NumItems();
    CList::IsEmpty();
    int InList(void* theItem);

private:
    friend class COrderedIterator;

    void ShiftLinks(CLink* theBaseLink);
};

class COrderedIterator : public CIterator
{
public:
    COrderedIterator(const COrderedList& theList);
    COrderedIterator(const COrderedList* theList);

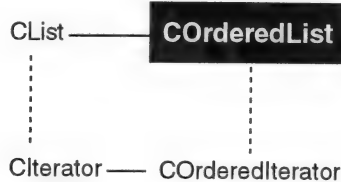
    int GetCurrentPosition(void) const;

    // Iterating methods:
    virtual void* Next(void);
    virtual void* Previous(void);

    virtual void* First(void);
    virtual void* Last(void);

private:
    CLink* itsCurrentPosLink;
};
#endif COrderedList_H
```

COrderedList



Description

COrderedList is a container class that can store references to any generic object. Since templates are not yet supported by every C++ compiler, objects are inserted via void pointers. Thus COrderedList stores objects by their address. It does not store "copies" of objects.

COrderedList is similar to CList, except that the items in the list have an order. A position is associated with each element of the list. The COrderedIterator iterates through the elements of the list in ascending position order.

Objects can be multiply inserted into the list. How objects are inserted into the same position on a list depends on whether the Insert or Replace method is used.

Heritage

Superclass: CList

Usage

Create an ordered list and then insert and remove items. Because of the use of void*, you must use casting in at least two places when using COrderedList under the current non-template implementation. Here is an example of use:

```

CList aList();
Foo* aFoo = new Foo(1,2,3);
aList.Insert((void*) aFoo); // Cast #1
Foo* fooItem;
CIterator doTo(aList);
while (fooItem = (aFoo*) doTo.Next()) // Cast #2
fooItem->Goo();
  
```

Data Members

None.

Friends

friend class COrderedIterator; class that iterates through the list

Public Methods

Constructor and Destructor Methods

COrderedList(void);

A constructor. It creates an empty list.

COrderedList(const COrderedList&theOtherList);

A copy constructor, which copies all of the attributes of theOtherList, including any pointers it contains. It does not copy the objects to which the list is pointing.

COrderedList& operator=(const COrderedList&theOtherList);

An assignment operator, which copies all of the attributes of theOtherList, including any pointers it contains. It does not copy the objects to which the list is pointing.

virtual ~COrderedList(void);

The destructor, which cleans up after the list but does not delete the objects to which the list is pointing.

List Operations

BOOLEAN Insert(void* theItem, int thePosition);

Inserts an item into a list. theItem is a void pointer to a generic object. In addition this method takes a position (thePosition) in the list in which you want to insert the item. When you iterate over the list, the items will be put in order. You thus do not have to worry about making the positions adjacent. For example, you can insert an item into position 1 and the next into position 38 and the next into position 1008. The positions are consecutive but do not need to be incremented by one each time, although you can do this if you desire.

When Insert puts an item into a position that already contains an item, it shifts the first item one position higher in the list. This method returns a Boolean value of TRUE if it has to do any shifting and FALSE if it does not.

```
void* Replace(void* theItem, int thePosition);
```

Inserts theItem into thePosition. If an item already occupies the given position, this method replaces the original item and returns it as a pointer. The void pointer is set to NULL if no item was replaced, and it points to an object if the item was replaced.

```
BOOLEAN Remove(void* theItem);
```

Removes the specified item (theItem) from the list and returns a Boolean value of TRUE. It returns FALSE if it does not remove the item, possibly because it could not find it.

```
void* Remove(int thePosition);
```

Removes the item at the given position (thePosition) and returns a pointer to that object. If it finds no item in that position, it returns NULL.

```
void* Find(int thePosition) const;
```

Returns a pointer to the item in the specified position (thePosition). It returns NULL if there is no item in that position.

```
void Renumber(int theBasePosition, int theIncrement);
```

Provides a way to reorder all of the items inside an ordered list. It takes a base position (theBasePosition) from which to start counting and the increment. For example, if the base position is 10 and the increment is 2, Renumber starts at position 10 and places the items that follow in positions 12, 14, 16, 18, and so on.

Protected Methods

None.

Private Methods

```
void ShiftLinks(CLink* theBaseLink);
```

An internal method that takes care of any shifting that needs to be done during an Insert operation.

Privately Inherited Methods Made Available

The following methods are privately inherited from CList. These methods are important. You cannot say that an ordered list is a type of CList because an ordered list does not inherit publicly from CList. The *only* CList methods available to COrderedList are the ones listed in this section.

CList::Clear;

CList::NumItems;

CList::IsEmpty;

int InList(void* theItem);

Returns an integer indicating how many times the specified item (theItem) was found in the list.

COrderedList.h

```
#ifndef COrderedList_H
#define COrderedList_H

#include "PwrNames.h"
#include "CLists.h"

class COrderedList : private CList
{
public:
    // Class Utility:
    COrderedList(void);
    COrderedList(const COrderedList& theOtherList);
    COrderedList& operator=(const COrderedList& theOtherList);

    // List Operations:
    BOOLEAN Insert(void* theItem, int thePosition);
    void* Replace(void* theItem, int thePosition);

    BOOLEAN Remove(void* theItem);
    void* Remove(int thePosition);

    void* Find(int thePosition) const;

    void Renumber(int theBasePosition, int theIncrement);

    // Privately inherited methods which are made available:
    CList::Clear;
    CList::NumItems;
    CList::IsEmpty;

    int InList(void* theItem);

private:
    friend class COrderedIterator;

    void ShiftLinks(CLink* theBaseLink);
};

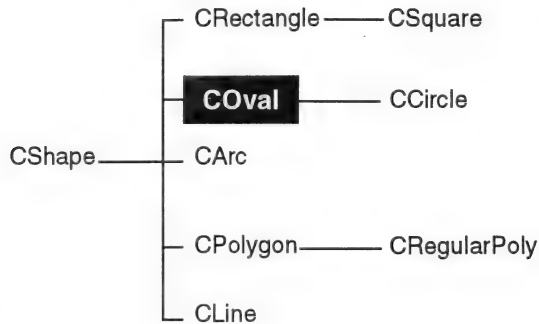
class COrderedIterator : public CIterator
{
public:
    COrderedIterator(const COrderedList& theList);
    COrderedIterator(const COrderedList* theList);

    int GetCurrentPosition(void) const;

    // Iterating methods:
    virtual void* Next(void);
    virtual void* Previous(void);
};
```

```
        virtual void* First(void);  
        virtual void* Last(void);  
    private:  
        CLink* itsCurrentPosLink;  
};  
#endif COrderedList_H
```

C Oval



Description

Paints a C Oval object inside a view.

Heritage

Superclass: CShape

Subclass: CCircle

Usage

Create a C Oval object and initialize it. Like all CShape classes, this class inherits all properties of CSubview, such as stickiness, enclosure, and so on.

Environment

The line of the oval is drawn with the pen; its interior is painted with the brush. You can set the color and pattern of both the pen and the brush. Also, you can set the width of the pen.

Data Members

None.

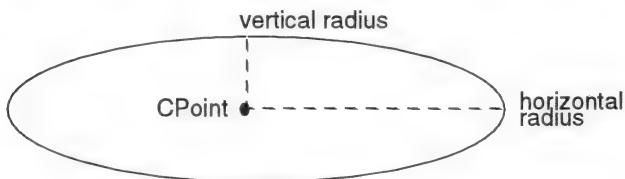
Public Methods

```

C Oval(CSubview* theEnclosure, const CPoint&
theCenter, UNITS theHRadius, UNITS
theVRadius);
  
```

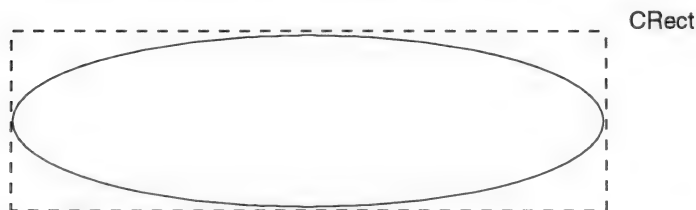
A constructor. theEnclosure is a pointer to the subview that will contain the icon. In addition, the constructor requires the

center point for the oval (a `CPoint` that is relative to the oval's enclosure), a horizontal radius, and a vertical radius.



```
COval(CSubview* theEnclosure, const CRect& theRegion);
```

A constructor. `theEnclosure` is a pointer to the subview that will contain the oval. `theRegion` is a coordinate location, local to the enclosure, that is used to place the oval.



```
COval(const COval& theOval);
```

A copy constructor that creates a new `COval` object with the same enclosure, color, visibility attributes, enabled/disabled attributes, environment, and so on as the original `COval` object. However, any views nested within the original `COval` object are not copied.

```
COval& operator=(const COval& theOval);
```

An assignment operator. It copies the attributes of the original `COval` object, creating a new `COval` object that has the same color, glue, environment, visibility state, and so on. However, any views nested within the original `COval` object are not copied.

```
virtual ~COval(void);
```

The destructor. It cleans up after the `COval` object and deletes any views nested within it.

```
BOOLEAN IOval(BOOLEAN isVisible, GLUETYPE theGlue);
```

The initializer, which allows the visibility status of the shape to be set and can take a glue type.


```
virtual void Draw(const CRect&
    theClippingRegion);
```

Takes care of any drawing the oval must do. theClippingRegion is the part of the oval that needs to be drawn, and it is in global, window-relative coordinates. If it is set to NULL, the entire oval is redrawn.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
```

```
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
```

```
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CSubview

```
virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,   BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
```

```

virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CShape

None.

Summary: COval.h

```

#ifndef COval_H
#define COval_H

#include "CShape.h"

class COval : public CShape
{
public:
    COval(CSubview* theEnclosure, const CPoint& theCenter,
          UNITS theHRadius, UNITS theVRadius);
    COval(CSubview* theEnclosure, const CRect& theRegion);
    COval(const COval& theOval);
    COval& operator=(const COval& theOval);
    virtual ~COval(void);

    BOOLEAN IOval(BOOLEAN isVisible, GLUETYPE theGlue);
    virtual void Draw(const CRect& theClippingRegion);
};

#endif COval_H

```

CPoint

CString

CRect

CPoint

Description

CPoint objects are data structures that specify locations on the screen. A CPoint is not a visible object; it is a mathematical entity. A CPoint object consists of two values, a horizontal (x) value and a vertical (y) value. Thus, each CPoint is an x,y coordinate. The functionality of this class is extensive. Every XVT-Power++ object has a CPoint object that tells it which origin to draw from and what its coordinates are relative to.

Heritage

Superclass: none

Usage

Create a CPoint object that has a horizontal and a vertical position, as follows:

```
CPoint a(1,5); CPoint b = a;  
PNT point; point.h = 2; point.v = 4; CPoint c(point);
```

Public Data Members

None.

Protected Data Members

None.

Private Data Members

UNITS	fH;	horizontal position
UNITS	fV;	vertical position

Public Methods

Constructor and Destructor Methods

CPoint(void);

A constructor. It takes no parameter and initializes the point with zero horizontal and zero vertical positions, (0,0).

CPoint(UNITS theHorizontalPos, UNITS theVerticalPos);

A constructor. Given a horizontal position (theHorizontalPos) and a vertical position (theVerticalPos) for initialization.

Methods for Setting a Point

CPoint& SetPoint(UNITS theHorizontalPos, UNITS theVerticalPos);

Sets a CPoint by giving it a new horizontal position (theHorizontalPos) and a new vertical position (theVerticalPos). It returns a reference to the object itself.

Member Operations

UNITS H(void) const;

Returns a number of logical units representing the CPoint's horizontal position.

void H(UNITS theHorizontalPos);

Given a horizontal position for the CPoint (theHorizontalPos), changes that position to this new value.

UNITS V(void) const;

Returns a number of logical units representing the CPoint's vertical position.

void V(UNITS theVerticalPos);

Given a vertical position for the CPoint (theVerticalPos), changes that position to this new value.

Conversion Operators

The following two conversion methods exist for purposes of compatibility with the XVT Portability Toolkit, making CPoints and PNTs completely interchangeable.

CPoint(PNT point);

Given an XVT Portability Toolkit PNT, converts it to a CPoint. The XVT Portability Toolkit's PNT structure is defined in the *XVT Programmer's Guide* as follows:

```
typedef struct {    /* mathematical point */
    short v;        /* vertical (y) coordinate */
    short h;        /* horizontal (x) coordinate */
} PNT;
```

operator PNT(void) const;

Converts a CPoint to an XVT Portability Toolkit PNT.

Binary Operators

CPoint& operator=(const CPoint& aCPoint);

Makes two CPoint objects equal.

CPoint& operator+=(const CPoint& aCPoint);

Adds the given CPoint to the current one. The two horizontal values are added and the two vertical values are added. For example, a CPoint with the value 5,5 added to one with the value 3,2 would yield a value of 8,7.

CPoint& operator-=(const CPoint& CPoint);

Subtracts the given CPoint from the current one. The two horizontal values are subtracted, and the two vertical values are subtracted. For example, a CPoint with the value 3,2 subtracted from one with the value 5,5 would yield a value of 2,3.

friend CPoint operator+(const CPoint& leftPoint, const CPoint& rightPoint);

Adds the rightPoint to the leftPoint. Unlike operator+,, which actually changes the CPoint, this method makes a temporary copy of the CPoint as it is when changed.

friend CPoint operator-(const CPoint& leftPoint, const CPoint& rightPoint);

Subtracts the rightPoint from the leftPoint. Unlike operator-,, which actually changes the CPoint, this method makes a temporary copy of the CPoint as it is when changed.

BOOLEAN operator==(const CPoint& aCPoint) const;

Returns a Boolean value of TRUE if the given CPoint object (aCPoint) is equal in size to the current CPoint and FALSE if it is not.

BOOLEAN operator!=(const CPoint& aCPoint) const;

Returns a Boolean value of TRUE if the given CPoint object is not equal in size to the current CPoint and FALSE if they are equal.

Methods for Coordinate System Conversion

It is very easy in XVT-Power++ to convert global (window relative) coordinates to local coordinates and *vice versa* because XVT-Power++ provides the Localize and Globalize methods. You can go back and forth between global and local coordinates without having to do any calculations.

```
CPoint Translate(const CView* fromView, const CView* toView);
```

Translates a CPoint object's coordinates. It takes the view from which the CPoint is being translated (fromView) and the view to which it is being translated (toView).

```
CPoint GetTranslated(const CView* fromLocalView, const CView* toView) const;
```

Unlike Translate, which permanently changes the CPoint's coordinates, GetTranslated returns a temporary copy of the CPoint as it is when changed. It, too, takes the view from which the CPoint is being translated (fromLocalView) and the view to which it is being translated (toView).

```
CPoint Globalize(const CView* fromView);
```

Converts the CPoint object's coordinates to global, window-relative coordinates. It takes a pointer to the view from which the globalizing is done (fromView).

```
CPoint GetGlobal(const CView* fromView) const;
```

Like Globalize, this method converts the CPoint object's coordinates to global, window-relative coordinates. It takes a pointer to the view from which the globalizing is done (fromView). Unlike Globalize, which actually changes the CPoint, GetGlobal returns a temporary copy of the CPoint as it is when changed.

```
CPoint Localize(const CView* toView);
```

Assumes that the CPoint is in global, window-relative coordinates and localizes the CPoint to the view toView.

```
CPoint GetLocal(const CView* toView) const;
```

Like Localize, this method localizes a CPoint object's coordinates to a view. Unlike Localize, which actually changes

the CPoint, GetLocal returns a temporary copy of the CPoint as it is when changed.

Summary: CPoint.h

```
#ifndef CPoint_H
#define CPoint_H

#include "PwrNames.h"
#include "xvt.h"
#include "CMem.h"

class CView;

class CPoint
{
public:
    // Constructors and destructor
    CPoint(void);
    CPoint(UNITS theHorizontalPos, UNITS theVerticalPos);

    // Setting Functions:
    CPoint& SetPoint(UNITS theHorizontalPos, UNITS theVerticalPos);

    // Member Operations:
    UNITS H(void) const;
    void H(UNITS theHorizontalPos);
    UNITS V(void) const;
    void V(UNITS theVerticalPos);

    // Conversion Operators:
    CPoint(PNT point);
    operator PNT(void) const;

    // Operators:
    CPoint& operator=(const CPoint& aCPoint);
    CPoint& operator+=(const CPoint& aCPoint);
    CPoint& operator-=(const CPoint& aCPoint);

    friend CPoint operator+(const CPoint& leftPoint,
                           const CPoint& rightPoint);
    friend CPoint operator-(const CPoint& leftPoint,
                           const CPoint& rightPoint);

    BOOLEAN operator==(const CPoint& aCPoint) const;
    BOOLEAN operator!=(const CPoint& aCPoint) const;

    // Coordinate system conversions
    CPoint Translate(const CView* fromView, const CView* toView);
    CPoint GetTranslated(const CView* fromLocalView,
                       const CView* toView) const;

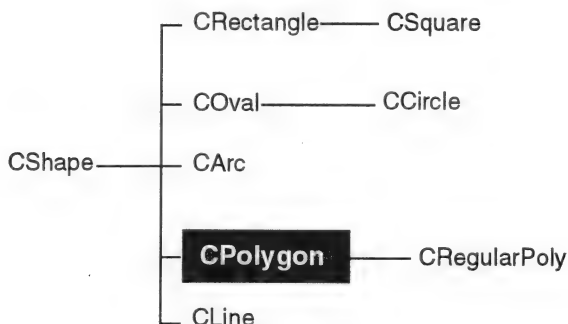
    CPoint Globalize(const CView* fromView);
    CPoint GetGlobal(const CView* fromView) const;

    CPoint Localize(const CView* toView);
    CPoint GetLocal(const CView* toView) const;

private:
    UNITS fh; // horizontal pixel
    UNITS fv; // vertical pixel
};

#endif CPoint_H
```

CPolygon



Description

CPolygon objects paint a polygon inside a view.

Heritage

Superclass: CShape

Subclass: CRegularPoly

Usage

Create a CPolygon object and initialize it. Like all shapes, this class inherits all properties of CSubview, such as stickiness, enclosure, and so on.

CPolygon is initialized with an array of CPoints. Thus, the polygon is not necessarily regular (contrast with CRegularPoly). The object draws the polygon by connecting the dots, or CPoints. If you choose to let the object fill its interior, then the last point will connect with the first.

Environment

The line of the polygon is drawn with the pen; its interior is painted with the brush. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width.

Public Data Members

None.

Protected Data Members

PNT*	itsPoints;	the polygon's points
int	itsNumberOfPoints;	the number of vertices+1

Private Data Members

BOOLEAN	itIsFilled;	whether the polygon has a fill
---------	-------------	--------------------------------

Public Methods

Constructor, Destructor, and Initializer Methods

```
CPolygon(CSubview* theEnclosure, const
        COrderedList* theListOfPoints);
```

A constructor. theEnclosure is a pointer to the subview that will contain the polygon. When you pass it an ordered list of points, this constructor connects these points into a polygon.

```
CPolygon(CSubview* theEnclosure, const CPoint*
        theArrayOfPoints, int theNumberOfPoints);
```

A constructor. theEnclosure is a pointer to the subview that will contain the polygon. When you pass it an array of points and give the number of points, this constructor connects these points into a polygon. This constructor, which uses an array, is an alternative to the preceding constructor, which uses an ordered list of points.

```
CPolygon(const CPolygon& thePolygon);
```

A copy constructor that creates a new CPolygon object with the same enclosure, color, visibility attributes, enabled/disabled attributes, environment, and so on as the original CPolygon object. However, any views nested within the original CPolygon object are not copied.

```
CPolygon& operator=(const CPolygon&
        thePolygon);
```

An assignment operator. It copies the attributes of the original CPolygon object, creating a new CPolygon object that has the same color, glue, environment, visibility state, and so on. However, any views nested within the original CPolygon object are not copied.

```
virtual ~CPolygon(void);
```

The destructor. It cleans up after the polygon and deletes any views nested within it.

```

BOOLEAN IPolygon(BOOLEAN isFilled= FALSE,
                  BOOLEAN isVisible    = TRUE,
                  GLUETYPE theGlue     = NULLSTICKY);

```

The initializer. By default, the CPolygon object has no interior fill, is visible, and uses the glue type NULLSTICKY. One or more of these defaults can be overridden.

```

virtual void SetFilled(BOOLEAN isFilled);

```

Takes a Boolean value; if TRUE, the polygon has a fill.

```

virtual BOOLEAN IsFilled(void) const;

```

Returns a Boolean value that indicates whether the polygon has a fill.

Inherited Methods

```

virtual void Draw(const CRect&
                  theClippingRegion);

```

Takes care of any drawing the polygon must do. theClippingRegion is the part of the polygon that needs to be drawn, and it is in global, window-relative coordinates. If it is set to NULL, the entire polygon is drawn.

```

virtual void Size(const CRect& theNewSize);

```

Sizes the polygon according to the coordinates of theNewSize. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, theNewSize, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated.

```

virtual void SetOrigin(const CPoint&
                       theChange);

```

Takes the current origin of a polygon and adds theChange to it. For example, suppose a view has an origin of 10, 20 and you want to shift the origin by three pixels to the right. You would set the origin by giving it a delta point of 3, 0, which shifts the view three pixels horizontally.

Protected Methods

```

CPolygon(CSubview* theEnclosure, const CRect&
          theRegion);

```

A constructor. theEnclosure is a pointer to the subview that will contain the polygon. theRegion is a coordinate location, local to the enclosure, that is used to place the polygon. This is

a protected constructor for classes such as CRegularPoly, which create their internal point structures themselves.

void AdjustInternalPoints(void);

An internal method that adjusts the internal points used to draw a polygon, relative to the object's region and origin.

Private Methods

CRect CreatePoints(const CPoint* theArrayOfPoints, int theNumberOfPoints);

Given a pointer to an array of CPoints and an integer value indicating the number of points, this method translates the points into coordinates for the XVT Portability Toolkit.

CRect CreatePoints(const CIdOrderedList* theListOfPoints);

Given a pointer to an ordered list of CPoints, this method translates the points into coordinates for the XVT Portability Toolkit.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
```

```
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
```

```
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CSubview

```
virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
```

```

virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CShape

None.

Summary: CPolygon.h

```

#ifndef CPolygon_H
#define CPolygon_H

#include "CShape.h"

class COrderedList;

class CPolygon : public CShape
{
public:
    // construct, destruct, init
    CPolygon(CSubview* theEnclosure, const COrderedList* theListOfPoints);
    CPolygon(CSubview* theEnclosure, const CPoint* theArrayOfPoints,
        int theNumberOfPoints);
    CPolygon(const CPolygon& thePolygon);
    CPolygon& operator=(const CPolygon& thePolygon);
    virtual ~CPolygon(void);

```



```

    BOOLEAN IPolygon(BOOLEAN isFilled = FALSE,
                     BOOLEAN isVisible = FALSE,
                     GLUETYPE theGlue = NULLSTICKY);

    virtual void Draw(const CRect& theClippingRegion);
    virtual void Size(const CRect& theNewSize);

    virtual void SetFilled(BOOLEAN isFilled);
    virtual BOOLEAN IsFilled(void) const;

    //inherited methods
    virtual void SetOrigin(const CPoint& theChange);

protected:
    PNT* itsPoints; // the polygon's points
    int itsNumberOfPoints; // the number of verteces+1

    CPolygon(CSubview* theEnclosure, const CRect& theRegion);
    void AdjustInternalPoints(void);

private:
    BOOLEAN itIsFilled; // wants an interior drawn

    CRect CreatePoints(const CPoint* theArrayOfPoints,
                      int theNumberOfPoints);
    CRect CreatePoints(const CIdOrderedList* theListOfPoints);
};

#endif CPolygon_H

```

CPrintMgr

CMem	CGlue	CSwitchBoard
CError	CDesktop	CStartup
	CEnvironment	CGlobalClassLib
	CGlobal	CGlobalUser
		CResourceMgr

CPrintMgr

Description

CPrintMgr receives printing task messages and takes care of printing for an XVT-Power++ application. Currently, only views can be printed. The print manager has a queue of printing jobs and contains information about the page(s) to be designated for each job. CPrintMgr thus includes a set of printing queue methods.

Heritage

Superclass: None

Usage

CPrintMgr is used internally, but there are times when you may want to access it directly. You can access a global instance of this class through the CGlobalClassLib object (G). CPrintMgr includes a set of print queue methods and a set of printing methods that are used when you have finished filling the queue and want to send the job to the printer. The DoPrint methods in the CApplication, CDocument, and CView classes already take care of all the print queue and print message tasks internally. You do not need to deal with the print manager at all unless you want to do customized printing that is not accommodated in the default DoPrint methods. If you are interested in implementing your own printing mechanisms, refer to the chapter on printing in the *XVT Programmer's Guide*.

Public Data Members

None.

Protected Data Members

<code>CList</code>	<code>itsPrintQueue;</code>	the view to print
<code>PRINT_RCD*</code>	<code>itsXVTPrintRecord;</code>	the XVT Portability Toolkit printing record
<code>const CString</code>	<code>*itsTitle;</code>	the title of the print job
<code>int</code>	<code>itsLastPage;</code>	the last page of output
<code>static WINDOW</code>	<code>itsPrintWindow;</code>	the XVT Portability Toolkit printing window

Private Data Members

None.

Constructor and Destructor Methods

`CPrintMgr(void);`

A constructor. It sets up some printing facilities for the application.

`virtual ~CPrintMgr(void);`

The destructor. It deletes the PrintMgr object when the application terminates.

Printing Queue Methods

`virtual void Insert(const CView* theView, CRect& theRegion, int thePage = LASTPage);`

Inserts a view into the printing queue. `theView` is the view to be inserted. `theRegion` is the clipping region, which is relative to the enclosure for the view. Finally, `thePage` is the page of the queue on which the view is to be printed. By default, a newly inserted view is printed on the last page. Then the next view is inserted on the page after the last page, which becomes the last page, and so on. In short, each new object is inserted at the end of the page queue and a page is added each time a view is inserted. You can insert a view into the queue several times if you want to print it on several different pages.

`virtual int Remove(const CView* theView, int thePage = ALLPages);`

Removes a view from the printing queue. `theView` is the view to be removed. `thePage` is useful if the view is printed on more than one page. This parameter allows you to specify particular

pages from which the view is to be removed. By default, the view is removed from all pages on which it appears.

```
virtual const CList GetQueue(int thePage = ALLPages) const;
```

Returns a list of all views that are in the printing queue. thePage allows you to get a list of everything that is printed on a particular page.

```
virtual void ClearQueue(void);
```

Removes everything from the printing queue.

Printing Methods

```
virtual BOOLEAN DoPrint(PRINT_RCD* printRecord,  
const CString& theTitle = NULLString);
```

A method that is called once the queue is full and the job is ready to be sent to the printer. DoPrint sets up a special method, which is a static method called PrintThread. See the description of this method in the “Protected Methods” section below. printRecord describes the print set-up, and theTitle is the title of the printing job. You can override this method to specify a different kind of printing or printing set-up.

```
static WINDOW GetPrintWindow(void);
```

Returns a value of NULL when no printing is being done, but if a printing window is being used, this method returns that window. GetPrintWindow is especially useful when you want to draw to the print manager’s print window. XVT-Power++ already takes care of this task internally, so you will probably never have to call GetPrintWindow.

Protected Methods

```
static BOOLEAN PrintThread(long theData);
```

Takes care of all printing. It is called independently of the rest of the application so that in some systems the printing can be spawned off and done while the application keeps running. PrintThread searches the queue and generates all printing events, creates the print window, and basically takes care of all of the actual interface with the XVT Portability Toolkit for printing. theData can be a long or a pointer to any kind of data that is needed during printing.

Private Methods

None.

Overrides

You can override both printing methods to change the printing implementation. Take care to set the global CPrintMgr object by calling G->SetPrintMgr.

Summary: CPrintMgr.h

```
#ifndef CPrintMgr_H
#define CPrintMgr_H

#include "PwrNames.h"
#include "xvt.h"
#include "CRect.h"

class CView;
class CString;

#define ALLPages -1
#define LASTPage -2
#define NEWPage -3

class CPrintMgr
{
public:
    CPrintMgr(void);
    virtual ~CPrintMgr(void);

    // Printing Queue Methods:
    virtual void Insert(const CView* theView,
                       CRect& theRegion,
                       int thePage = LASTPage);

    virtual int Remove(const CView* theView, int thePage = ALLPages);
    virtual const CList GetQueue(int thePage = ALLPages) const;
    virtual void ClearQueue(void);

    // Printing Methods:
    virtual BOOLEAN DoPrint(PRINT_RCD* printRecord,
                           const CString& theTitle = NULLString);

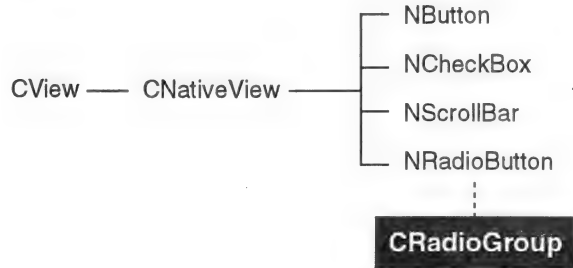
    static WINDOW GetPrintWindow(void);

protected:
    CList itsPrintQueue; // The view to print
    PRINT_RCD* itsXVTPrintRecord; // The the XVT printing record
    const CString* itsTitle; // The print document title
    int itsLastPage; // The last page of output

    static WINDOW itsPrintWindow; // The XVT printing window
    static BOOLEAN PrintThread(Long theData);
};

#endif // CPrintMgr_H
```

CRadioGroup



Description

CRadioGroup is a class that groups radio buttons. By definition, radio buttons must be grouped together. Unlike check boxes, which allow the user to select multiple options by checking more than one box at a time, radio buttons within a group can be selected only one at a time. Selecting one radio button means to deselect another.

Heritage

Superclass: CNativeView

Usage

You cannot instantiate radio buttons directly. Instead, you create a group and then add radio buttons to it. The group's size is automatically increased as new buttons are added to it.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

COrderedList	*itsButtons;	the list of grouped buttons
int	itsSelectedId;	ID of selected button
int	itsBorderSpace;	spacing between buttons and grouper

Public Methods

```
CRadioGroup(CSubview *theEnclosure, const  
CPoint& theTopLeft);
```

A constructor. theEnclosure is a pointer to the subview that will contain the radio group. theTopLeft is a coordinate location, local to the enclosure, that is used to place the group.

```
CRadioGroup(const CRadioGroup& theGroup);
```

A copy constructor. It copies all of the radio buttons contained within the group.

```
CRadioGroup& operator=(const CRadioGroup&  
theGroup);
```

An assignment operator. It copies all of the radio buttons contained within the group.

```
virtual ~CRadioGroup(void);
```

The destructor. It cleans up after the group and deletes all of the buttons it contains.

```
BOOLEAN IRadioGroup(const CString& theTitle= NULLString,  
                     BOOLEAN isVisible      = TRUE,  
                     GLUETYPE theGlue      = NULLSTICKY);
```

The initializer. It takes a title, a visibility state, and a glue type. The title consists of a string that is displayed beneath the group.

Grouper Utilities

```
virtual int AddButton(const CPoint &aPoint,  
                     const CString& theButtonText= NULLString,  
                     long theCommand           = NULLcmd,  
                     long theAttributes        = NULL,  
                     GLUETYPE theGlue         = NULLSTICKY);
```

One of two methods that allow you to add buttons to a radio group. It provides the means to add one button at a time. AddButton takes a coordinate where the button will be located (&aPoint); a title, which is the string of text that appears beside the button; the number of the command that is generated when the button is selected; and a glue type. theAttributes takes a value from a set of the XVT Portability Toolkit-provided attributes that you can give to native views. You can OR together the appropriate control flag constants into an attribute value. Refer to the table in the section on CNativeView for a listing of the possible control flags for theAttributes. This table appears in the description of the CNativeView constructor.

AddButton returns an integer, which is the ID number of the radio button. This ID number is important; many other methods on this class take radio button IDs as parameters.

```
virtual int AddButtons(int theFirstResourceId,
                      int theNumberOfButtons,
                      int theFirstCommand,
                      long theAttributes      = NULL,
                      int theSelectedButton   = NULL,
                      UNITS theSeparation     = 0,
                      DIRECTION theDirection = VERTICAL);
```

One of two methods that allow you to add buttons to a radio group. It provides the means to add several buttons at a time. It adds the buttons using a set of string resources for the titles of the buttons.

theFirstResourceId is the ID number of the string resource associated with the first button in the group; ID numbers for the strings associated with other buttons created in the group are assigned successively, based on this number.

theNumberOfButtons specifies the total number of buttons you want to add to the group. If the number of buttons is, say, five, the resources are traversed from the first resource all the way through four more.

theFirstCommand is the number of the command that is generated when the first button is selected. Each time a new button in the group is created, this number is incremented by one. theAttributes takes a value from a set of the XVT Portability Toolkit-provided attributes that you can give to native views. You can OR together the appropriate control flag constants into an attribute value. Refer to the table in the section on CNativeView for a listing of the possible control flags for theAttributes. This table appears in the description of the CNativeView constructor.

theSelectedButton is a number ranging from one all the way through the total number of buttons inserted into the group. It is the number of the button that you want to be selected upon creation. theSeparation and theDirection both pertain to the actual placement of the buttons. The buttons can be placed either vertically or horizontally (direction), and you specify the number of pixels that separate each button from the next.

AddButtons returns the ID number of the button corresponding to the first resource ID. The IDs of the remaining buttons created follow sequentially.

Example:

Note that in the following example, the string resources are numbered in *descending* consecutive order. For portability reasons, string resources cannot be placed in an ascending consecutive order inside the URL file. If you are defining resources for the Macintosh, for instance, a string list resource will be created instead of the the usual string resource, and this is not what is desired.

Step 1: Define radio button titles as string resources in the URL file:

```
STRING 212 "Hello"
STRING 211 "Bonjour"
STRING 210 "Hola"
```

Step 2: Create a radio group and add buttons as follows:

```
CSubview* enc;
CPoint point;

CRadioGroup *group = new CRadioGroup(enc, point);
group->AddButtons(210, // STRING ID of first button
                 3,   // number of buttons
                 10,  // command for first button
                 NULL, // XVT attributes
                 2,   // the selected button
                 5,   // separation between buttons
                 HORIZONTAL); // layout
```

```
virtual void RemoveButton(int theButtonId);
```

Given the ID number of a button, removes the button from the group.

```
virtual void SetSelectedButton(int id);
```

Given the ID number of a button, makes this the selected button in the group.

```
virtual int GetSelectedButton(void) const;
```

Returns the ID number of the button in the group that is currently selected.

```
virtual void Draw(const CRect&
                 theClippingRegion);
```

An overridden method that takes care of any drawing the radio group must do. Basically, it draws the background and foreground colors specified by the group's environment. theClippingRegion is the portion of the group that needs to be drawn, and it is in global, window-relative coordinates. If it is set to NULL, the entire group is drawn.

Border Space for Button Placement

```
virtual void SetBorderSpace(int thePixelSpace);
```

Sets the border space (thePixelSpace) between the grouper and the buttons it contains.

```
virtual int GetBorderSpace(void) const;
```

Returns an integer indicating the number of pixels between the grouper and the buttons it contains.

Protected Methods

None.

Private Methods

```
int AddButtonInternal(const CPoint &aPoint,
    const CString& theButtonText = NULLString,
    long theCommand              = NULLcmd,
    long theControlAttributes    = NULL,
    GLUTYPE theGlue              = NULLSTICKY);
```

An internal method that creates radio buttons and inserts them into a group.

Summary: NRadioButton.h

Note: Both **CRadioGroup** and **NRadioButton** share the same header file. This file is shown in the discussion of each class.

```
#ifndef NRadioButton_H
#define NRadioButton_H

#ifdef STDH
#include "CNativeView.h"
#include "CSubview.h"
#endif STDH
#ifdef DOS
#include "CNativWw.h"
#include "CSubview.h"
#endif

class COrderedList;

class CRadioGroup : public CSubview
{
public:
    CRadioGroup(CSubview *theEnclosure, const CPoint& theTopLeft);
    CRadioGroup(const CRadioGroup& theGroup);
    CRadioGroup& operator=(const CRadioGroup& theGroup);
    virtual ~CRadioGroup(void);

    BOOLEAN IRadioGroup(const CString& theTitle = NULLString,
        BOOLEAN isVisible = TRUE,
        GLUTYPE theGlue = NULLSTICKY);
```

```

// Grouper utility:
virtual int AddButton(const CPoint &aPoint,
    const CString& theButtonText = NULLString,
    long theCommand = NULLcmd,
    long theAttributes = NULL,
    GLUTYPE theGlue = NULLSTICKY);

virtual int AddButtons(int theFirstResoureId,
    int theNumberOfButtons,
    int theFirstCommand,
    long theAttributes = NULL,
    int theSelectedButton = NULL,
    UNITS theSeparation = 0,
    DIRECTION theDirection = VERTICAL);

virtual void RemoveButton(int theButtonId);

virtual void SetSelectedButton(int id);
virtual int GetSelectedButton(void) const;

virtual void Draw(const CRect& theClipping);

//Border space for button placement:
virtual void SetBorderSpace(int thePixelSpace);
virtual int GetBorderSpace(void) const;

private:
    COrderedList *itsButtons; // the list of grouped buttons
    int itsSelectedId; // is of selected button
    int itsBorderSpace; // spacing between buttons and grouper

    intAddButtonInternal(const CPoint &aPoint,
        const CString& theButtonText = NULLString,
        long theCommand = NULLcmd,
        long theAttributes = NULL,
        GLUTYPE theGlue = NULLSTICKY);
};

class CRadioButton : public CNativeView
{
    friend class CRadioGroup;

public:
    // Inherited utility:
    virtual void DoHit(CONTROL_INFO controlInfo);
    virtual void SetSizing(BOOLEAN isSizable);
    virtual void SetTitle(const CString& theNewTitle);

protected:
    CRadioGroup *itsRadioGroup;

    NRadioButton(CSubview *theEnclosure,
        const CRect& theRegion,
        CRadioGroup* theRadioGroup,
        long theAttributes = NULL,
        const CString& theTitle = NULLString);
    NRadioButton(const CRadioButton& theButton);
    NRadioButton& operator=(const CRadioButton& theButton);
    ~NRadioButton(void);

    BOOLEAN IRadioButton(const CString& theTitle = NULLString,
        BOOLEAN isEnabled = TRUE,
        long theCommand = NULLcmd,
        BOOLEAN isVisible = TRUE,
        long theGlue = NULLSTICKY);

```

```
};  
#endif NRadioButton_H
```

CRect

CString

CPoint

CRect

CUnits

Description

CRect is a data structure that allows you to store a rectangular region (set of coordinates) on the screen. A CRect is not a visible object. That is, the dimensions of a CRect define an area in which another object will be located. Every XVT-Power++ object has a CRect that tells it where to draw. A CRect object consists of four measurements—a top, a left, a bottom, and a right—that define its rectangular region.

Heritage

Superclass: none

Usage

Create a CRect object and initialize it with top-left and bottom-right coordinates.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

UNITS	itsLeft;	the left side
UNITS	itsTop;	the top side
UNITS	itsRight;	the right side
UNITS	itsBottom;	the bottom side

Public Methods

Constructor Methods

CRect(void);

A constructor. It creates a rectangle that has a top, left, bottom, and right value of zero (0).

CRect(UNITS theLeft, UNITS theTop, UNITS theRight, UNITS theBottom);

A constructor. It takes four units values that set the size of the rectangle: theLeft, theTop, theRight, and theBottom.

CRect(const CPoint& theTopLeftCorner, const CPoint& theBottomRightCorner);

A constructor. It instantiates a CRect based on CPoint coordinates for theTopLeftCorner and theBottomRightCorner.

Resetting Methods

CRect& SetRect(UNITS theLeft, UNITS theTop, UNITS theRight, UNITS theBottom);

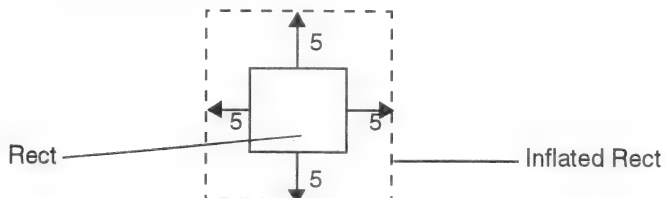
Once a CRect object has been constructed, this method resets its dimensions with a new set of left, top, right, and bottom values.

CRect& SetRect(CPoint theTopLeftCorner, CPoint theBottomRightCorner);

Once a CRect object has been constructed, this method resets its dimensions, taking CPoint coordinates for theTopLeftCorner and theBottomRightCorner

CRect& Inflate(UNITS theInflation);

Increases the dimensions of a CRect in all directions by the number of units given for theInflation. For example, if theInflation is 5, this method makes the CRect bigger by five units in all directions. A negative number decreases the size of the CRect. A value of -5 would decrease the size of the CRect by 5 units in all directions.



CRect& Inflate(UNITS theHInflation, UNITS theVInflation);

Increases the dimensions of a CRect in horizontally and vertically by the number of logical units given. The logical units must be of the same type for the horizontal and vertical directions. For example, if you specify a vertical inflation of two inches, then the horizontal inflation must also be specified in some number of inches. For a detailed explanation of the term “logical units,” see CUnits.

Side Operations

Each of the following methods performs an operation on one side of a CRect. Notice that these methods are overloaded.

UNITS Left(void) const;

Returns the number of logical units but takes no argument. It simply returns a value representing the coordinates of the CRect’s left side.

UNITS Left(UNITS theNewLeft);

Resets the coordinates of the CRect’s left side, according to the value of theNewLeft, and returns the new value.

UNITS Top(void) const;

Returns the number of logical units representing the coordinates of the CRect’s top side.

UNITS Top(UNITS theNewTop);

Resets the coordinates of the CRect’s top side, according to the value of theNewTop, and returns the new value.

UNITS Right(void) const;

Returns the number of logical units representing the coordinates of the CRect’s right side.

UNITS Right(UNITS theNewRight);

Resets the size of the CRect’s right side, according to the value of theNewRight, and returns the new value.

UNITS Bottom(void) const;

Returns the number of logical units representing the coordinates of the CRect’s bottom side.

UNITS Bottom(UNITS theNewBottom);

Resets the coordinates of the CRect's bottom side, according to the value of theNewBottom, and returns the new value.

Height and Width Operations

UNITS Height(void) const;

Returns the number of units representing the height of the CRect.

UNITS Height(UNITS theNewHeight);

Sets the height of the CRect according to the value of theNewHeight.

UNITS Width(void) const;

Returns the number of logical units representing the width of the CRect.

UNITS Width(UNITS theNewWidth);

Sets the width of the CRect according to the value of theNewWidth.

Utility CRect Operations

BOOLEAN IsPointInRect(const CPoint& thePoint) const;

Given a CPoint coordinate, this method returns TRUE or FALSE depending on whether that point is contained within the CRect's coordinates.

BOOLEAN IsEmpty(void) const;

Returns TRUE if the CRect is empty, that is, if all four points equal the same thing.

CRect GetInflatedRect(UNITS theInflation) const;

Returns a temporary value, theInflation, representing a CRect that has been inflated. The actual CRect through which this method is invoked is not modified. GetInflatedRect simply takes a picture of this CRect in its inflated state and returns it as a temporary value.

CRect GetInflatedRect(UNITS theHInflation, UNITS theVInflation) const;

Returns two temporary values, theHInflation and theVInflation, representing a CRect that has been inflated horizontally and vertically. The actual CRect through which this method is invoked is not modified. GetInflatedRect simply

takes a picture of this CRect in its inflated state and returns it as as two temporary values.

Type Conversion Utilities

The following two conversion methods exist for purposes of compatibility with the XVT Portability Toolkit, making CRects and RCTs completely interchangeable.

CRect(const RCT& rect);

Creates a CRect through the XVT Portability Toolkit's RCT structure, which is defined in the *XVT Programmer's Guide* as follows:

```
typedef struct {           /* mathematical rectangle */
    short top;             /* top coordinate */
    short left;            /* left coordinate */
    short bottom;          /* bottom coordinate */
    short right;           /* right coordinate */
} RCT;
```

operator RCT(void) const;

A conversion operator that creates an XVT Portability Toolkit RCT from a CRect.

Binary Operators

CRect& operator=(const CRect& aCRect);

Operator equal makes two CRects the same size.

BOOLEAN operator==(const CRect& aCRect) const;

Returns a Boolean value of TRUE if the given CRect object (aCRect) is equal in size and location to the current CRect and FALSE if it is not.

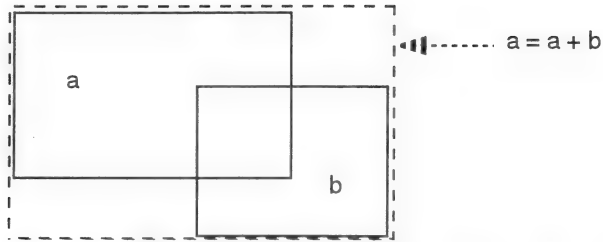
BOOLEAN operator!=(const CRect& aCRect) const;

Returns a Boolean value of TRUE if the given CRect object is not equal in size and location to the current CRect and FALSE if they are equal.

Union Operators

CRect& operator+=(const CRect& aCRect);

Operator plus-equal is a union type of operation, which, when a CRect is given, has the effect of resizing the CRect, as follows:



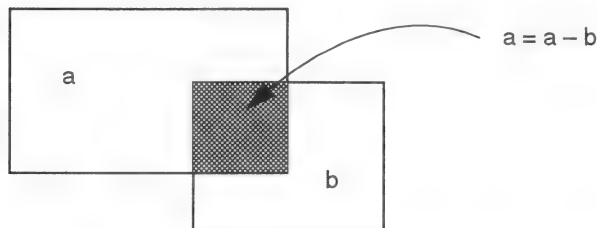
```
friend CRect operator+(const CRect& leftRect,
const CRect& rightRect);
```

Adds two CRect objects, leftRect and rightRect, returning a union of those objects. Unlike the operator+=, which actually modifies the CRect, this method simply adds the rectangles and returns a temporary copy.

Intersection Operators

```
CRect& operator-=(const CRect& aCRect);
```

Subtracts two CRect objects, returning the intersecting rectangle, as follows:



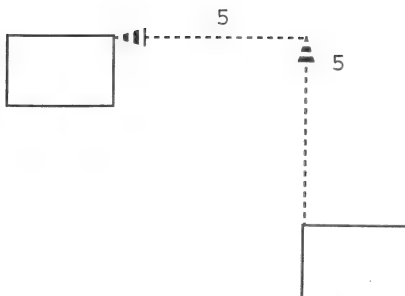
```
friend CRect operator-(const CRect& leftRect,
const CRect& rightRect);
```

Subtracts two CRect objects, returning an intersection of those objects. Unlike the operator-=, which actually modifies the CRect, this method simply subtracts the rectangles and returns a temporary copy.

Translation Operators

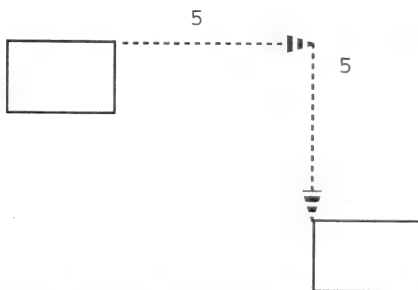
```
CRect& operator-=(const CPoint& aPoint);
```

Subtracts a CPoint (aPoint) from a CRect, which is equal to a translation. This operation moves the CRect as follows:



```
CRect& operator+=(const CPoint& aPoint);
```

Adds a CPoint (aPoint) to a CRect, which is equal to a translation. This operation moves the CRect as follows:



```
friend CRect operator+(const CRect& theRect, const CPoint& thePoint);
```

Adds a CPoint to a CRect object, returning a translation as shown above.

```
friend CRect operator+(const CPoint& thePoint, const CRect& theRect);
```

Adds a CRect to a CPoint (thePoint), which is equal to a translation as shown above.

```
friend CRect operator-(const CRect& theRect, const CPoint& thePoint);
```

Subtracts a CPoint from a CRect, which is equal to a translation as shown above.

Methods for Coordinate System Conversion

It is very easy in XVT-Power++ to convert global to local coordinates and *vice versa* because XVT-Power++ provides the Localize and Globalize methods. You can go back and forth

between global and local coordinates without having to do any calculations.

CRect Translate(const CView* fromView, const CView* toView);

Translates a CRect object's coordinates. It takes the view from which the CRect is being translated (fromView) and the view to which it is being translated (toView).

CRect GetTranslated(const CView* fromLocalView, const CView* toView) const;

Unlike Translate, which permanently changes the CRect's coordinates, GetTranslated returns a temporary copy of the CRect as it is when changed. It, too, takes the view from which the CRect is being translated (fromLocalView) and the view to which it is being translated (toView).

CRect Globalize(const CView* fromView);

Converts the CRect object's coordinates to global, window-relative coordinates. It takes a pointer to the view from which the globalizing is done (fromView).

CRect GetGlobal(const CView* fromView) const;

Like Globalize, this method converts the CRect object's coordinates to global, window-relative coordinates. It takes a pointer to the view from which the globalizing is done (fromView). Unlike Globalize, which actually changes the CRect, GetGlobal returns a temporary copy of the CRect as it is when changed.

CRect Localize(const CView* toView);

Assumes that the CRect is in global, window-relative coordinates and localizes the CRect to the view toView.

CRect GetLocal(const CView* toView) const;

Like Localize, this method localizes a CRect object's coordinates to a view. Unlike Localize, which actually changes the CRect, GetLocal returns a temporary copy of the CRect as it is when changed.

Summary: CRect.h

```
#ifndef CRect_H
#define CRect_H
```

```

#include "PwrNames.h"
#include "CMem.h"
#include "CPoint.h"
#include "xvt.h"

class CView;

class CRect
{
public:
    // Constructors
    CRect(void);
    CRect(UNITS theLeft, UNITS theTop, UNITS theRight,
          UNITS theBottom);
    CRect(const CPoint& theTopLeftCorner,
          const CPoint& theBottomRightCorner);

    // CRect Resetting Functions
    CRect& SetRect(UNITS theLeft, UNITS theTop, UNITS theRight,
                  UNITS theBottom);
    CRect& SetRect(CPoint theTopLeftCorner,
                  CPoint theBottomRightCorner);
    CRect& Inflate(UNITS theInflation);
    CRect& Inflate(UNITS theHInflation, UNITS theVInflation);

    // Side Operations
    UNITS Left(void) const;
    UNITS Left(UNITS theNewLeft);
    UNITS Top(void) const;
    UNITS Top(UNITS theNewTop);
    UNITS Right(void) const;
    UNITS Right( UNITS theNewRight);
    UNITS Bottom(void) const;
    UNITS Bottom(UNITS theNewBottom);

    // Height and Width Operations
    UNITS Height(void) const;
    UNITS Height(UNITS theNewHeight);
    UNITS Width(void) const;
    UNITS Width(UNITS theNewWidth);

    // Utility CRect Operations
    BOOLEAN IsPointInRect(const CPoint& thePoint) const;
    BOOLEAN IsEmpty(void) const;
    CRect GetInflatedRect(UNITS theInflation) const;
    CRect GetInflatedRect(UNITS theHInflation,
                          UNITS theVInflation) const;

    // Type Conversion Utilities
    CRect(const RCT& rect);
    operator RCT(void) const;

    // Binary Operators
    CRect& operator=(const CRect& aCRect);
    CRect& operator+=(const CRect& aCRect);
    CRect& operator+=(const CPoint& aPoint);
    CRect& operator-=(const CRect& aCRect);
    CRect& operator-=(const CPoint& aPoint);

    BOOLEAN operator==(const CRect& aCRect) const;
    BOOLEAN operator!=(const CRect& aCRect) const;

```

```
friend CRect operator+(const CRect& leftRect,
    const CRect& rightRect);
friend CRect operator-(const CRect& leftRect,
    const CRect& rightRect);
friend CRect operator+(const CRect& theRect,
    const CPoint& thePoint);
friend CRect operator+(const CPoint& thePoint,
    const CRect& theRect);
friend CRect operator-(const CRect& theRect,
    const CPoint& thePoint);
friend CRect operator-(CPoint thePoint, const CRect& theRect);

// Coordinate system conversions
CRect Translate(const CView* fromView, const CView* toView);
CRect GetTranslated(const CView* fromLocalView,
    const CView* toView) const;

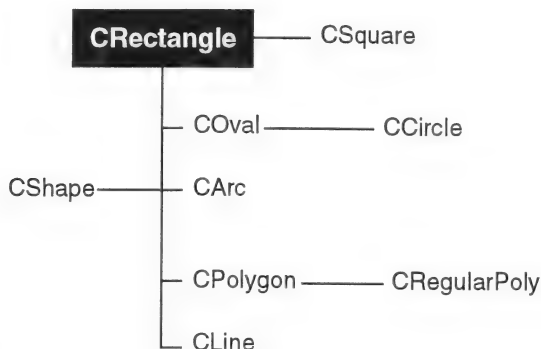
CRect Globalize(const CView* fromView);
CRect GetGlobal(const CView* fromView) const;

CRect Localize(const CView* toView);
CRect GetLocal(const CView* toView) const;

private:
    UNITS itsLeft;
    UNITS itsTop;
    UNITS itsRight;
    UNITS itsBottom;
};

#endif CRect_H
```

CRectangle



Description

CRectangle objects paint a rectangle inside a CView or CSubview.

Heritage

Superclass: CShape

Subclass: CSquare

Usage

Create a CRectangle object and initialize it. Like all shapes, this class inherits all properties of CSubview, such as stickiness, enclosure, and so on.

To give the rectangle rounded corners, initialize it with `isCornerRounded = TRUE` and provide the desired corner width and height.

Environment

The lines in the rectangle are drawn with the *pen*; its interior is painted with the *brush*. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width.

Public Data Members

None.

Protected Data Members

None.

Protected Data Members

None.

Private Data Members

BOOLEAN	itHasRoundedCorners	whether the rectangle has rounded corners
int	itsCornerWidth	width of the rounded corners, in pixels
int	itsCornerHeight	height of the rounded corners, in pixels

Public Methods

Constructor, Destructor, and Initialization Methods

CRectangle(CSubview* theEnclosure, const CRect& theRegion);

The constructor. It takes a pointer to an enclosure and a region that is a CRect. The enclosure is the subview that contains the CRectangle object; all CShape objects must be placed within an enclosure. theRegion is a coordinate location that is local to the enclosure; this region indicates where the new CRectangle object is to be placed.

CRectangle(const CRectangle& theRectangle);

A copy constructor that creates a new CRectangle object with the same enclosure, color, glue, visibility attributes, enabled/disabled attributes, environment, and so on as the original CRectangle object. However, any shapes nested within the original CRectangle object are not copied.

CRectangle& operator=(const CRectangle& theRectangle);

An assignment operator. It copies the attributes of the original CRectangle object, creating a new CRectangle object that has the same color, glue, environment, visibility state, and so on. However, any shapes nested within the original CRectangle object are not copied.

virtual ~CRectangle(void);

The destructor. It cleans up after the CRectangle object and deletes any views nested within it.

**BOOLEAN IRectangle(BOOLEAN hasRoundCorners = FALSE,
UNITS theCornerWidth = 0,
UNITS theCornerHeight = 0,**


```

    BOOLEAN isVisible      = TRUE,
    GLUETYPE theGlue      = NULLSTICKY);

```

The initializer. Rectangles can optionally have rounded corners. If this is desired, set the Boolean parameter `hasRoundCorners` to `TRUE`. If this parameter is set to `TRUE`, then you must also set the `theCornerWidth` and `theCornerHeight` parameters, which take numbers indicating, in pixels, how high and deep the rounding be. This initializer also allows the visibility status and the glue type of the rectangle to be set.

```

virtual void Draw(const CRect&
theClippingRegion);

```

Takes care of any drawing the rectangle must do. `theClippingRegion` is the part of the rectangle that needs to be drawn, and it is in global, window-relative coordinates. If it is set to `NULL`, the entire rectangle is drawn.

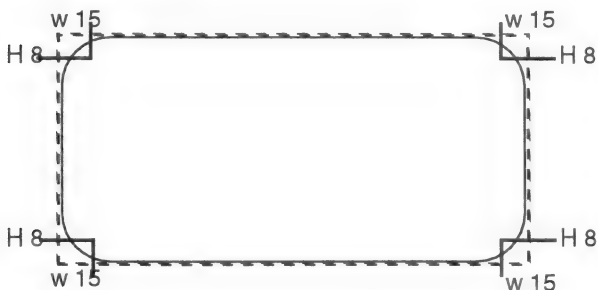
Utilities

```

virtual void SetRoundedCorners(BOOLEAN
isCornerRounded, UNITS theCornerWidth=SAME,
UNITS theCornerHeight=SAME);

```

Rectangles can optionally have rounded corners. If this is desired, set the Boolean parameter `hasRoundCorners` to `TRUE`. If this parameter is set to `TRUE`, then you must also set the `theCornerWidth` and `theCornerHeight` parameters, which take numbers, in logical units, indicating how high and how deep the rounding should be. The default value, `SAME`, retains the values that are already set.



Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);  
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);  
virtual CUnits* GetUnits(void) const;  
virtual void SetUnits (CUnits* theCoordinateUnits);  
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);  
virtual void Deactivate(void);  
virtual void Disable(void);  
virtual void DoCommand(long theCommand, void* theData=NULL);  
virtual BOOLEAN DoPrint(const CRect& theRegion) const;  
virtual void DoTimer(long theTimerId);  
virtual void DoUser(long theUserId, void* theData);  
virtual void Draw(void);  
virtual void Enable(void);  
virtual CView* FindHitView(const CPoint& theLocation) const;  
virtual CRect GetClippedFrame(void) const;  
virtual long GetCommand(void) const;  
virtual long GetDoubleCommand(void) const;  
virtual CSubview* GetEnclosure(void);  
virtual const CEnvironment* GetEnvironment(void) const;  
virtual CRect GetFrame(void) const;  
virtual CRect GetGlobalFrame(void) const;  
virtual CPoint GetGlobalOrigin(void) const;  
virtual GLUETYPE GetGlue(void) const;  
virtual int GetId(void) const;  
virtual CRect GetLocalFrame(void) const;  
virtual CPoint GetOrigin(void) const;  
virtual const CString GetTitle(void) const;  
virtual CWindow* GetCWindow(void) const;  
virtual CWireFrame* GetWireFrame(void) const;  
virtual void Glue();  
virtual void Hide(void);
```

```
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CSubview

```
virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
```

```
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation,CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
```

```
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);
```

From CShape

None.

Summary: CRectangle.h

```
#ifndef CRectangle_H
#define CRectangle_H

#include "CShape.h"

class CRectangle : public CShape
{
public:
    // construct, destruct, init
    CRectangle(CSubview* theEnclosure, const CRect& theRegion);
    CRectangle(const CRectangle& theRectangle);
    CRectangle& operator=(const CRectangle& theRectangle);
    virtual ~CRectangle(void);

    BOOLEAN IRectangle(BOOLEAN hasRoundCorner          = FALSE,
                      UNITS theCornerWidth            = 0,
                      UNITS theCornerHeight           = 0,
                      BOOLEAN isVisible               = TRUE,
                      GLUETYPE theGlue                 = NULLSTICKY);

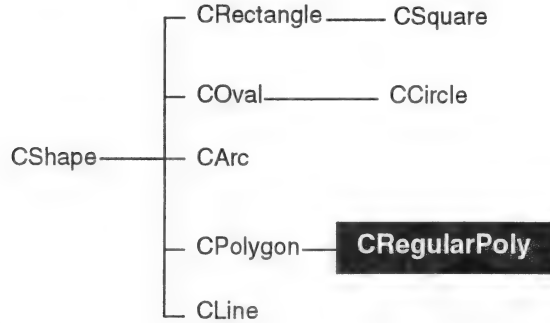
    virtual void Draw(const CRect& theClippingRegion);

    // util:
    virtual void SetRoundedCorners(BOOLEAN isCornerRounded,
                                   UNITS theCornerWidth=SAME,
                                   UNITS theCornerHeight=SAME);

private:
    BOOLEAN itHasRoundCorners; // draw rectangle with rounded corners
    UNITS itsCornerWidth;      // width and height of rounded corners
    UNITS itsCornerHeight;     // ...
};

#endif CRectangle_H
```

CRegularPoly



Description

CRegularPoly objects paint a regular polygon inside a CSubview.

Heritage

Superclass: CPolygon

Usage

Create a CRegularPoly object and initialize it. Like all shapes, this class inherits all properties of CSubview, such as stickiness, enclosure, and so on.

Environment

The lines in the regular polygon are drawn with the *pen*; its interior is painted with the *brush*. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

float	itsStartingAngle;	drawing reference angle
UNITS	itsRadius;	distance from center to

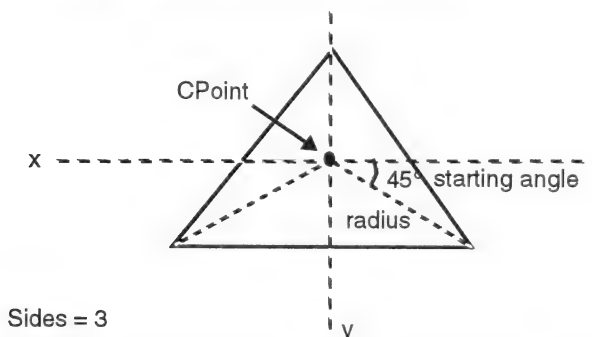
CPoint	*itsCenter;	a vertex the center of the regular polygon
--------	-------------	--

Public Methods

Constructor, Destructor, and Initializer Methods

```
CRegularPoly(CSubview* theEnclosure, const
CPoint& theCenterPoint, UNITS theRadius, int
theNumberOfSides, float theStartingAngle =
0.0);
```

A constructor. theEnclosure is a pointer to the subview that will contain the regular polygon. In addition, the constructor requires the center point for the regular polygon (a CPoint that is relative to the regular polygon's enclosure) and the radius of the regular polygon. Also, it takes an integer value that specifies the number of sides for the polygon. The minimum number of sides is three. Finally, it takes a degree measure for the starting angle of the polygon, as illustrated here:



```
CRegularPoly(const CRegularPoly& thePolygon);
```

A copy constructor that creates a new CRegularPoly object with the same enclosure, color, visibility attributes, enabled/disabled attributes, environment, and so on as the original CRegularPoly object. However, any shapes nested within the original CRegularPoly object are not copied.

```
CRegularPoly& operator=(const CRegularPoly&
thePolygon);
```

An assignment operator. It copies the attributes of the original CRegularPoly object, creating a new CRegularPoly object that has the same color, glue, environment, visibility state, and so

on. However, any views nested within the original CRegularPoly object are not copied.

```
virtual ~CRegularPoly(void);
```

The destructor. It cleans up after the regular polygon and deletes any views nested within it.

```
BOOLEAN IRegularPoly(int theNumberOfSides,  
                      BOOLEAN isInteriorDrawn= FALSE,  
                      BOOLEAN isVisible      = TRUE,  
                      long theGlue          = NULLSTICKY);
```

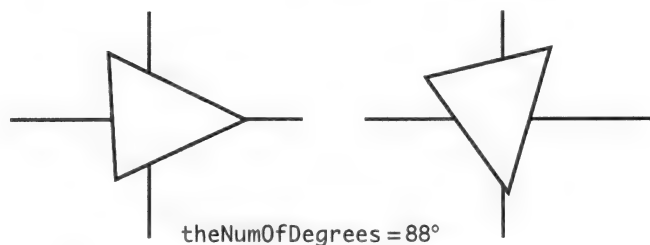
The initializer. It takes an integer value that specifies the number of sides for the polygon. By default, the CRegularPoly object has no interior fill, is visible, and uses the glue type NULLSTICKY.

```
virtual void Size(const CRect& theNewSize);
```

Sizes the regular polygon according to the coordinates of theNewSize. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, theNewSize, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated.

```
virtual void Rotate(float theNumOfDegrees);
```

Rotates the polygon in a clockwise direction by the value of theNumOfDegrees.



```
virtual void SetNumberOfSides(int  
    theNumberOfSides);
```

Sets the number of sides for the regular polygon. The minimum number of sides is three.

```
virtual int GetNumberOfSides(void) const;
```

Returns an integer value indicating the number of sides of the regular polygon.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);  
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);  
virtual CUnits* GetUnits(void) const;  
virtual void SetUnits (CUnits* theCoordinateUnits);  
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);  
virtual void Deactivate(void);  
virtual void Disable(void);  
virtual void DoCommand(long theCommand, void* theData=NULL);  
virtual BOOLEAN DoPrint(const CRect& theRegion) const;  
virtual void DoTimer(long theTimerId);  
virtual void DoUser(long theUserId, void* theData);  
virtual void Draw(void);  
virtual void Enable(void);  
virtual CView* FindHitView(const CPoint& theLocation) const;  
virtual CRect GetClippedFrame(void) const;  
virtual long GetCommand(void) const;  
virtual long GetDoubleCommand(void) const;  
virtual CSubview* GetEnclosure(void);  
virtual const CEnvironment* GetEnvironment(void) const;  
virtual CWindow* GetCWindow(void) const;  
virtual CRect GetFrame(void) const;  
virtual CRect GetGlobalFrame(void) const;  
virtual CPoint GetGlobalOrigin(void) const;  
virtual GLUETYPE GetGlue(void) const;  
virtual int GetId(void) const;  
virtual CRect GetLocalFrame(void) const;  
virtual CPoint GetOrigin(void) const;  
virtual const CString GetTitle(void) const;
```

```

virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0,
    BOOLEAN isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,
    BOOLEAN isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,
    BOOLEAN isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0,
    BOOLEAN isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
    isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);

```

```
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubObjbects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
```

```

virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CShape

None.

From CPolygon

```

void AdjustInternalPoints(void);
CRect CreatePoints(const CPoint* theArrayOfPoints, int theNumberOfPoints);
CRect CreatePoints(const COrderedList* theListOfPoints);
virtual void Draw(const CRect& theClippingRegion);
virtual BOOLEAN IsFilled(void) const;
virtual void SetFilled(BOOLEAN isFilled);
virtual void SetOrigin(const CPoint& theChange);

```

Summary: CRegularPoly.h

```

#ifndef CRegularPoly_H
#define CRegularPoly_H

#include "CPolygon.h"

class CRegularPoly : public CPolygon
{
public:
    // construct, destruct , init
    CRegularPoly(CSubview* theEnclosure,const CPoint& theCenterPoint,
        UNITS theRadius, int theNumberOfSides);
    CRegularPoly(const CRegularPoly& thePolygon);
    CRegularPoly& operator=(const CRegularPoly& thePolygon);
    virtual ~CRegularPoly(void);

    BOOLEAN IRegularPoly(int theNumberOfSides,
        BOOLEAN isInteriorDrawn      = FALSE,
        BOOLEAN isVisible             = TRUE,
        long theGlue                  = NULLSTICKY);

    virtual void Size(const CRect& theNewSize);

    virtual void SetNumberOfSides(int theNumberOfSides);
    virtual int GetNumberOfSides(void) const;

private:
    float itsStartingAngle;    // drawing reference angle
    UNITS itsRadius;           // distance from center to a vertex
    CPoint *itsCenter;         // the center of the Regular Polygon

    void CalculatePoints(void);
};

```

```
#endif CRegularPoly_H
```

CResourceMgr

CMem	CGlue	CSwitchBoard
CError	CDesktop	CStartup
	CEnvironment	CGlobalClassLib
	CGlobal	CGlobalUser
		CResourceMgr
		CPrintMgr

Description

Currently, CResourceMgr is a file that takes care of icon resources under Motif and Open Look platforms. Normally, all the resources, including icons, can be defined through the XVT Portability Toolkit's URL. However, under X platforms, icon resources are integrated in a different way. The main thing that the CResourceMgr does is give instructions, along with examples, on how to create icon resources for the X platforms.

Heritage

Superclass: none

Usage

See the specific instructions for each platform in the summary section. Any UNIX application that uses icon resources must link in to the resource manager, whether it is the one defined here by XVT-Power++ or a user-defined resource manager. Be sure to define the platform at compile time.

Summary: CResourceMgr.h

```

////////////////////////////////////
// file: CResourceMgr.c
// author: JIT
// date : 5/13/91
// Resource management file: contains an example of how to create icon

```

```

// resources on some platforms.
////////////////////////////////////

#include "xvt.h"
#include "CGlobal.h"
////////////////////////////////////

// X RESOURCES:
// To build an X icon or cursor resource, simply follow the following
// simple four steps.
#if (defined WSMTF || defined WSXOL)
#define xvtxi_bld_icon bld_icon
#define xvtxi_bld_cursor bld_cursor
#define xvtx_resource_table_init resource_table_init

extern "C" {
#include "xsrman.h"
}

// STEP 1: Include the bitmap file:
#include "pwr.ico"

// STEP 2: Define an icon or cursor resource type variable:
static ICON_RESOURCE pwrIcon;

// STEP 3: Place resource into XVT's rousce information table:
RESOURCE_INFO rtable[] =
{
    { "ICON", NULLicon, (char *) &pwrIcon},
    { "", 0, (char *) NULL } // THIS MUST BE THE LAST LINE
};

// STEP 4: Build the resource:
RESOURCE_INFO* resource_table_init(void)
{
    bld_icon(&pwrIcon, NULLheight, NULLwidth, pwr_bits);
    return(rtable);
}

#endif

```

CScroller

CSubview — CVirtualFrame — **CScroller** — CListBox

Description

CScroller attaches scrollbars to the visible area of the virtual frame; it is a virtual frame with scrollbars. Objects nested inside a scroller scroll automatically as the user manipulates the scrollbars. When users drag or size objects inside a scroller, the contents scroll automatically as the object is dragged beyond the visible borders.

Heritage

Superclass: CVirtualFrame

Subclass: CListBox

Usage

Create a CScroller object and place objects inside it (see CVirtualFrame). You must also indicate the following:

- Whether vertical scrollbars are to be attached.
- Whether horizontal scrollbars are to be attached.
- Step size: number of units to scroll per scrolling event.
- Page size: number of units to scroll per thumb scrolling event.
- Whether thumbtracking is to be used: scroll contents *as the user drags* the thumb. If this feature is not desired, the contents are scrolled *upon release* of a thumb drag.

Environment

The scroller border is drawn with the *pen*; you can set the pen color, pattern, and width. The scroller background is drawn with the *brush*. The brush pattern has been disabled and is always PAT_SOLID, even if you give it another setting. You can define the brush *color*, however. The color of the scrollbars is system-defined.

Public Data Members

None.

Protected Data Members

NScrollBar	*itsHScrollBar;	horizontal scrollbar
NScrollBar	*itsVScrollBar;	vertical scrollbar

Private Data Members

BOOLEAN	itIsThumbtracking;	whether thumbtracking is supported
CList	itsContents;	a list of the scroller's subviews

Public Methods

Constructor, Destructor, and Initializer Methods

CScroller(CSubview* theEnclosure, const CRect& theRegion, UNITS theVirtualWidth=0, UNITS theVirtualHeight=0);

A constructor. A constructor. theEnclosure is a pointer to the subview that will contain the scroller. theRegion is a coordinate location, local to the enclosure, that is used to place the scroller. The parameters theVirtualWidth and theVirtualHeight together define the size of the *virtual* area enclosing the scroller.

CScroller(CWindow* theScrollableWindow, UNITS theVirtualWidth = 0, UNITS theVirtualHeight = 0);

A constructor. theScrollableWindow is a window with scrollbars that will contain the scroller. The scroller will occupy the entire window's client area and will be operated using the window's scrollbars. The parameters theVirtualWidth and theVirtualHeight together define the size of the *virtual* area enclosing the scroller.

CScroller(const CScroller& theScroller);

A copy constructor that duplicates the attributes, but not the contents, of a scroller. That is, it copies the scroller but does not do a deep copy of any views nested within it.

CScroller& operator=(const CScroller& theScroller);

An assignment operator that duplicates the attributes, but not the contents, of a scroller. That is, it copies the scroller but does not do a deep copy of any views nested within it.

~CScroller(void);

The destructor. It cleans up after the scroller and deletes any views nested within it.

**BOOLEANIScroller(BOOLEANhasHorizontalScrollBar= TRUE,
 BOOLEAN hasVerticalScrollBar = TRUE,
 BOOLEAN isThumbTracking = FALSE,
 UNITS theLineIncrement = 10,
 UNITS thePageIncrement = 50,
 BOOLEAN isVisible = TRUE,
 long theGlue = NULLSTICKY);**

The initializer. It allows you to specify whether the scroller has a horizontal scrollbar, whether it has a vertical scrollbar, and whether it dynamically tracks the scrollbar's thumb. `theLineIncrement` specifies the number of pixels to scroll for a line increment; similarly, `thePageIncrement` specifies the number of pixels to scroll for a page increment. Like all initializers for classes in the `CSubview` hierarchy, this initializer takes a visibility state and a glue type.

Scroller Utilities

**virtual void SetHIncrements(UNITS
 theLineIncrement, UNITS thePageIncrement);**

Sets the horizontal increments of the scroller, in units, for both line increments and page increments.

**virtual void SetVIncrements(UNITS
 theLineIncrement, UNITS thePageIncrement);**

Sets the vertical increments of the scroller, in units, for both line increments and page increments.

virtual UNITS GetHLineIncrement(void) const;

Returns the number of units that have been set for a horizontal line increment.

virtual UNITS GetVLineIncrement(void) const;

Returns the number of units that have been set for a vertical line increment.

virtual UNITS GetHPageIncrement(void) const;

Returns the number of units that have been set for a horizontal page increment.

virtual UNITS GetVPageIncrement(void) const;

Returns the number of units that have been set for a vertical page increment.

```
virtual void SetThumbtracking(BOOLEAN  
isThumbTracking);
```

Sets the thumbtracking for the scroller, taking a Boolean value of TRUE or FALSE.

```
virtual BOOLEAN IsThumbtracking(void) const;
```

Returns a Boolean value of TRUE if the scroller supports dynamic thumbtracking and FALSE if it does not.

Inherited Horizontal and Vertical Scrolling Methods

The following two methods are called when a scrollbar receives a scrolling event. The event propagates from the scrollbar to the CScroller object. theEventType designates the kind of scroll: line up, line down, page up, or page down (the terms “up” and “down” are used for both horizontal and vertical scrolling, though the effects are different for horizontal scrolling). theThumbPosition is the numerical value for placement of the thumb when either thumb repositioning or dynamic thumb tracking is used. This method responds to the scrollbar event by scrolling the views contained inside the scroller.

```
virtual void VScroll(SCROLL_CONTROL  
theEventType, UNITS theThumbPosition);
```

A method called when a scrollbar receives a vertical scroll event.

```
virtual void HScroll(SCROLL_CONTROL  
theEventType, UNITS theThumbPosition);
```

A method called when a scrollbar receives a horizontal scroll event.

Inherited Utilities

```
virtual void ShrinkToFit(void);
```

Shrinks the virtual region of the frame so that it is just large enough to fit all of the subviews contained in the virtual frame. The scroller’s scrollbars are updated as needed.

```
virtual void ScrollViews(void);
```

An overridden method that scrolls the views contained in the scroller.

```
virtual CPoint AutoScroll(UNITS  
horizontalChange, UNITS verticalChange);
```

Overrides the AutoScroll method on CView. The scroller receives this event when the user drags one of its subviews towards the outside of the scroller. The dragging produces an

autoscroll event because the subview is being dragged outside of its bounds. `theHorizontalChange` takes a number of pixels. If the number is zero (0), no autoscrolling occurs; if the number is positive, the view scrolls to the right; if it is negative, the view scrolls to the left. `theVerticalChange` also takes a number of pixels. If the number is zero (0), no autoscrolling occurs; if the number is positive, the view scrolls downward; if it is negative, the view scrolls upward.

virtual void DoSetDragging(BOOLEAN isDraggable);

Takes a Boolean value that sets the dragging of a scroller to TRUE so that the scroller can be dragged with the mouse or to FALSE so that the scroller cannot be dragged. It then recursively sets the dragging for the scroller's child views.

virtual void DoSetSizing(BOOLEAN isSizable);

Takes a Boolean value that sets the sizing of a scroller to TRUE so that the scroller can be sized or to FALSE so that the scroller cannot be sized. It then recursively sets the sizing for the scroller's child views.

virtual void AddSubview(const CView* theSubview);

An overridden method that adds a pointer to a CSubview object to the contents of the virtual frame. This method ensures that the virtual region of the frame is large enough to include the newly added object, sizing it when necessary.

virtual void DoCommand(long theCommand, void* theData=NULL);

CScroller overrides DoCommand to handle some internal XVT-Power++ commands.

virtual CPoint GetGlobalOrigin(void) const;

Returns the scroller's origin (starting point), relative to the window.

virtual CRect GetClippedFrame(void) const;

Views are clipped to their enclosures, so portions of a view may not be visible. This method returns a region (CRect) with the coordinates of the visible portion of a clipped view. Usually, you will call GetClippedFrame before you call DoDraw in order to pass the smallest region possible to DoDraw. Because Draw and DoDraw take window-relative coordinates, GetClippedFrame returns a window-relative CRect.

virtual void Size(const CRect& theRect);

Sizes the scroller according to the coordinates of theRect. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, theRect, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated.

virtual void DoDraw(const CRect& theClippingRegion);

The scroller refreshes (draws) itself and notifies each of its subviews to do the same. theClippingRegion is the part of the scroller that needs to be refreshed and is in global, window-relative coordinates; if it is set to NULL, the entire scroller is drawn. This method is automatically called after an E_UPDATE event.

virtual void Draw(void);

The scroller draws itself and notifies each of its subviews to do the same. By default the clipping region is set to the entire region that the scroller occupies. If you do not want to calculate a clipping region and just want to draw the entire scroller, then you can call this method without passing anything.

virtual const CList* GetSubviews(void) const;

Returns a pointer to the scroller's list of subviews.

Protected Methods

Inherited Methods

Both of the following methods are called automatically when the scrollbar setting needs to be changed.

virtual void SetHScrollRange(UNITS theLeft, UNITS theRight, UNITS thePosition);

Depending on the position of the scroller, this overridden method sets the range of horizontal scrolling by defining the left and right boundaries of the scrollbar. thePosition sets the position of the scrollbar thumb. Basically, SetHScrollRange updates the scrollbar by giving it a minimum and maximum position, as well as the position of the thumb.

virtual void SetVScrollRange(UNITS theTop, UNITS theBottom, UNITS thePosition);

Depending on the position of the scroller, this overridden method sets the range of vertical scrolling by defining the top and bottom boundaries of the scrollbar. thePosition sets the

position of the scrollbar thumb. Basically, `SetVScrollRange` updates the scrollbar by giving it a minimum and maximum position, as well as the position of the thumb.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Enable(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual long GetCommand(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWindow* GetCWindow(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Hide(void);
```

```

virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);

```

From CSubview

```

virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoEnable(void);

```

```

virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation,List* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CVirtualFrame

```

virtual void Draw(const CRect& theClippingRegion);
virtual void EnlargeToFit(const CRect& theRegionToInclude);
CPoint GetScrollingOrigin(void);
virtual CRect GetVirtualFrame(void) const;

```



```

virtual void PostSize(void);
virtual void PreSize(void);
void SetScrollingOrigin(const CPoint& theNewOrigin);
virtual void SetVirtualFrame(unsigned int theNewWidth, unsigned int theNewHeight);

```

Summary: CScroller.h

```

#ifndef CScroller_H
#define CScroller_H

#define SBWIDTH 15

#include "PwrDef.h"

#include CVirtualFrame_i
#include CWindow_i

class NScrollBar;

class CScroller :public CVirtualFrame
{
public:
    // Constructor, destructor, and initializer
    CScroller(CSubview* theEnclosure, const CRect& theRegion,
        UNITS theVirtualWidth=0, UNITS theVirtualHeight=0);

    CScroller(const CScroller& theScroller);
    CScroller& operator=(const CScroller& theScroller);
    ~CScroller(void);

    BOOLEAN IScroller(BOOLEAN hasHorizontalScrollBar = TRUE,
        BOOLEAN hasVerticalScrollBar = TRUE,
        BOOLEAN isThumbTracking = FALSE,
        UNITS theLineIncrement = 10,
        UNITS thePageIncrement = 50,
        BOOLEAN isVisible = TRUE,
        GLUETYPE theGlue = NULLSTICKY);

    // Scroller utility:
    virtual void SetHIncrements(UNITS theLineIncrement,
        UNITS thePageIncrement);
    virtual void SetVIncrements(UNITS theLineIncrement,
        UNITS thePageIncrement);

    virtual UNITS GetHLineIncrement(void) const;
    virtual UNITS GetVLineIncrement(void) const;
    virtual UNITS GetHPageIncrement(void) const;
    virtual UNITS GetVPageIncrement(void) const;

    virtual void SetThumbtracking(BOOLEAN isThumbTracking);
    virtual BOOLEAN IsThumbtracking(void) const;

```

```

    // Inherited Utility:
    virtual void ShrinkToFit(void);
    virtual void ScrollViews(const CPoint& theNewOrigin);
    virtual CPoint AutoScroll(UNITS horizontalChange,
        UNITS verticalChange);
    virtual void VScroll(SCROLL_CONTROL theEventType,
        UNITS theThumbPosition);
    virtual void HScroll(SCROLL_CONTROL theEventType,
        UNITS theThumbPosition);
    virtual void DoSetDragging(BOOLEAN isDraggable);
    virtual void DoSetSizing(BOOLEAN isSizable);
    virtual void AddSubview(const CView* theSubview);

    virtual void DoCommand(long theCommand, void* theData = NULL);
    virtual CPoint GetGlobalOrigin(void) const;
    virtual CRect GetClippedFrame(void) const;

    virtual void Size(const CRect& theNewSize);
    virtual const CList* GetSubviews(void) const;

protected:
    NScrollBar *itsHScrollBar; // Horizontal scrollbar
    NScrollBar *itsVScrollBar; // Vertical scrollbar

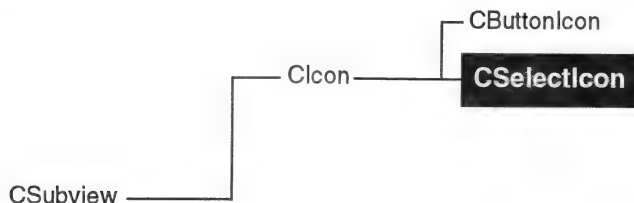
    virtual void SetHScrollRange(UNITS theLeft, UNITS theRight,
        UNITS thePosition);
    virtual void SetVScrollRange(UNITS theTop, UNITS theBottom,
        UNITS thePosition);

private:
    BOOLEAN itIsThumbtracking;
};

#endif CScroller_H

```

CSelectIcon



Description

CSelectIcon is a class that behaves like a selection box or a check box. When you click on a CSelectIcon object, it becomes selected and remains in its selected state until you click on it again. Like all select boxes, CSelectIcon sends a DoCommand to its enclosure when a selection or deselection occurs. For more information, see CIcon and CButtonIcon.

Heritage

Superclass: CIcon

Usage

Simply instantiate this class.

Environment

Under some platforms, the icon drawing appears in the foreground color, as does its title. Open spaces in the drawing appear in the background color. You can set the background and foreground colors. Under other platforms, the icon's color is fixed as defined. For portability, set the colors appropriately even if the information is not used.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

int	itsSelectedRID;	icon to use to represent selected state
long	itsDeselectCommand;	command generated upon deselection

Public Methods

Constructor, Destructor, and Initializer Methods

CSelectIcon(CSubview* theEnclosure, const CRect& theRegion);

A constructor. theEnclosure is a pointer to the subview that will contain the icon. theRegion is a coordinate location, local to the enclosure, that is used to place the icon.

CSelectIcon(const CSelectIcon& theIcon);

A copy constructor that causes two icons to draw the same picture. As with all subviews, only the top-level view is copied. Any views nested within the icon are not copied.

CSelectIcon& operator=(const CSelectIcon& theIcon);

An assignment operator that causes two icons to draw the same picture. As with all subviews, only the top-level view is copied. Any views nested within the icon are not copied.

virtual ~CSelectIcon(void);

The destructor. It cleans up after the icon and deletes any views nested within it.

```

BOOLEAN ISelectIcon(int theEnabledRID      = NULLicon,
                     int theDisabledRID    = NULLicon,
                     int theSelectedRID    = NULLicon,
                     const CString theTitle = NULLString,
                     long theSelectCommand = NULLcmd,
                     long theDeselectCommand = NULLcmd,
                     BOOLEAN isVisible    = TRUE,
                     GLUETYPE theGlue      = NULLSTICKY);

```

The initializer. Like the initializer of CIcon, it takes a resource ID for the enabled and disabled states of the icon, a visibility state, and a glue type. Also like the CIcon initializer, it takes a theTitle parameter, which is the title of the icon. This title is displayed underneath the icon and centered. In addition to these parameters, CSelectIcon has a theSelectedRID setting the specifies the resource or the picture that is used to represent the icon in its *selected* state. It also has two different commands that

are passed in at the initializer level: a command that it generates in its selected state (`theSelectCommand`) and a command that it generates when it becomes deselected (`theDeselectCommand`).

virtual BOOLEAN IsSelected(void) const;

Returns a Boolean value indicating whether the icon is in a selected state (TRUE) or a deselected state (FALSE).

virtual void Deselect(void);

A method that causes the icon to become deselected, changing the icon's visual representation.

virtual void Select(void);

A method that causes the icon to become selected.

Mouse Event Methods

CSelectIcon overrides two mouse methods to implement its selection behavior. Mouse events are passed directly to the select icon at the location where the event occurs. Each of the following methods has a `theButton` argument that specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right). By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button.

**virtual void MouseDouble(CPoint theLocation,
 short theButton = 0,
 BOOLEAN isShiftKey = FALSE,
 BOOLEAN isControlKey = FALSE);**

When the user double clicks a mouse button over a select icon containing local coordinate `theLocation`, the icon receives and handles this event by changing into selected mode. A select `DoCommand` is sent.

**virtual voidMouseDown(CPoint theLocation,
 short theButton = 0,
 BOOLEAN isShiftKey = FALSE,
 BOOLEAN isControlKey = FALSE);**

When the user presses down the left mouse button over a select icon containing local coordinate `theLocation`, the icon receives and handles this event by toggling between selected mode and deselected mode. The appropriate `DoCommand` is sent.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWindow* GetCWindow(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
```

```

virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

```

```

virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoPrintDraw(const CRect& theClippingRegion);

virtual void DoSetDragging(BOOLEAN isDraggable);

virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);

virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);

virtual void DoSetGlue(GLUETYPE theGlue);

virtual void DoSetOrigin(const CPoint& theDeltaPoint);

virtual void DoSetSizing(BOOLEAN isSizable);

virtual void DoShow(void);

virtual void DoSize(const CRect& theNewSize);

virtual CView* FindDeepSubview(const CPoint& theLocation) const;

virtual FindEventTarget(const CPoint& theLocation) const;

virtual CView* FindSubview(int theViewId) const;

virtual CView* FindSubview(const CPoint& theLocation) const;

virtual void FindSubviews(const CPoint& theLocation,List* theList) const;

CView* GetKeyFocus(void) const;

int GetNumViews(void) const;

virtual CView* GetSelectedView(void) const;

virtual const CList* GetSubObjs(void) const = NULL;

virtual const CList* GetSubviews(void) const;

virtual void PlaceBottomSubview(const CView* theView);

virtual void PlaceTopSubview(const CView* theView);

virtual void RemoveSubview(const CView* theSubview);

virtual void SetKeyFocus(CView *theFocusedView);

virtual void SetSelectedView(CView* theSelectedView);

```

From Icon

```

virtual void Disable(void);

virtual void Draw(const CRect& theClippingRegion);

virtual void Enable(void);

virtual void Invert(void);

virtual void SetFont(const FONT& theFont, BOOLEAN isUpdate = FALSE);

```



```
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theTitle);
virtual void Size(const CRect& theNewRegion);
```

Overrides

You should override `Validate` and `Key` to handle user-defined validations.

Summary: CSelectIcon.h

```
#ifndef CSelectIcon_H
#define CSelectIcon_H

#include "CIcon.h"

class CSelectIcon : public CIcon
{
public:
    // Construct, Destruct, Initialize:
    CSelectIcon(CSubview* theEnclosure, const CRect& theRegion);
    CSelectIcon(const CSelectIcon& theIcon);
    CSelectIcon& operator=(const CSelectIcon& theIcon);
    virtual ~CSelectIcon(void);

    BOOLEAN ISelectIcon(int theEnabledRID    = NULLicon,
                       int theDisabledRID    = NULLicon,
                       int theSelectedRID    = NULLicon,
                       const CString theTitle = NULLString,
                       long theSelectCommand = NULLcmd,
                       long theDeselectCommand = NULLcmd,
                       BOOLEAN isVisible     = TRUE,
                       GLUETYPE theGlue     = NULLSTICKY);

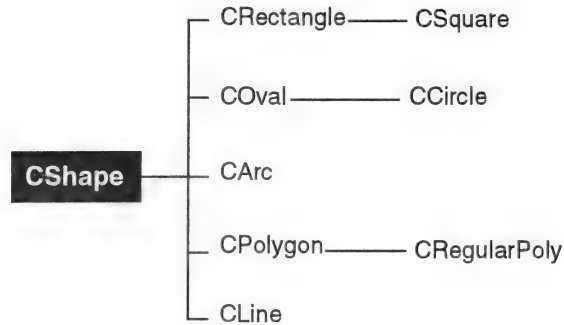
    virtual BOOLEAN IsSelected(void) const;
    virtual void Deselect(void);
    virtual void Select(void);

    // View utilities:
    virtual void MouseDouble(CPoint theLocation, short theButton,
                            BOOLEAN isShiftKey, BOOLEAN isControlKey);
    virtual void MouseDown(CPoint theLocation, short theButton,
                           BOOLEAN isShiftKey, BOOLEAN isControlKey);

private:
    int itsSelectedRID;           // icon to use to represent selected state
    long itsDeselectCommand;     // command generated upon deselection
};

#endif CSelectIcon_H
```

CShape



Description

CShape is an abstract parent class for such drawing tools as rectangles, circles, lines, polygons, and so on. This is a subclass of CSubview, so CShape objects can contain other nested views, and they can be moved and sized.

Heritage

Superclass: CSubview

Subclasses: CRectangle, COval, CArc, CPolygon, CLine

Usage

This class is an abstraction for drawing tools. You should use it to derive other shape classes as needed.

Data Members

None.

Public Methods

None.

Protected Methods

```
CShape(CSubview *theEnclosure, const CRect& theRegion);
```

The constructor. It is identical to the constructor of CView and CSubview. It takes a pointer to an enclosure and a region that is a CRect. The enclosure is the subview that contains the CShape

object; all CShape objects must be placed within an enclosure. theRegion is a coordinate location that is local to the enclosure; this region indicates where the new CShape object is to be placed.

CShape(const CShape& theShape);

A copy constructor that creates a new CShape object with the same enclosure, color, visibility attributes, enabled/disabled attributes, environment, and so on as the original CShape object. However, any shapes nested within the original CShape object are not copied.

CShape& operator=(const CShape& theShape);

An assignment operator. It copies the attributes of the original CShape object, creating a new view that has the same color, glue, environment, visibility state, and so on. However, any shapes nested within the original CShape object are not copied.

virtual ~CShape(void);

The destructor.

BOOLEAN IShape(BOOLEAN isVisible = TRUE, GLUETYPE theGlue = NULLSTICKY);

The initializer, which requires that the visibility status of the shape be set and that it be given a glue type.

virtual void Draw(const CRect& theClippingRegion);

Sets up the environment for the shape—the colors, the pens, the brushes, and whatever attributes the shape will have. This method must be overridden by derived shapes, yet it should also be called. That is, the overridden methods should call the inherited Draw so that the environment is set.

theClippingRegion is the part of the shape that needs to be drawn, and it is in global, window-relative coordinates.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

virtual void ChangeFont(FONT theFont, FONT_PART theChange);

virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);

virtual CUnits* GetUnits(void) const;

virtual void SetUnits (CUnits* theCoordinateUnits);

```
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUTYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWindow* GetCWindow(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
```

```

virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);

```

```
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation,CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);
```

Overrides

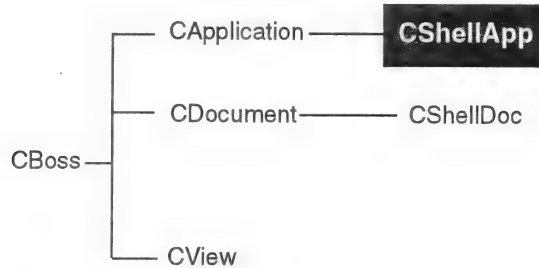
Each inherited class must override the CShape draw function and call it before proceeding with its own drawing code. You may also have to override some methods inherited from CSubview, such as dragging and sizing.

Summary: CShape.h

```
#ifndef CShape_H
#define CShape_H
#include "CSubview.h"

class CShape : public CSubview
{
    protected:
        // counstruct, init, destruct
        CShape(CSubview *theEnclosure, const CRect& theRegion);
        CShape(const CShape& theShape);
        CShape& operator=(const CShape& theShape);
        virtual ~CShape(void);
        virtual void Draw(const CRect& theClippingRegion);
        BOOLEAN IShape(BOOLEAN isVisible = TRUE,  GLUETYPE theGlue = NULLSTICKY);
};
#endif CShape_H
```

CShellApp



Description

CShellApp is an example of how to override the minimal set of methods of CApplication. It initializes the inherited application and creates a shell document.

Heritage

Superclass: CApplication

Usage

Rename this class and add methods as necessary—or simply use it as an example. This class should be instantiated by main as illustrated by CStartup.cxx. CShellApp is the basis for the NewProject tool provided with XVT-Power++ on some platforms. Consult your installation guide for information on how to execute this program.

Public Methods

CShellApp(void);

The constructor, which calls DoNew. It tells the application to DoNew as if the user had selected “New” from the File menu for the very first time. A new document is automatically created.

virtual void StartUp(void);

An example of how to override the StartUp method of CApplication.


```
virtual void DoCommand(long theCommand, void*
    theData=NULL);
```

An example of how a CApplication-derived class can override the DoCommand method. The void pointer theData can be used to pass any user-defined structure to the DoCommand method.

```
virtual BOOLEAN DoNew(void);
```

Creates a shell document and then gives it a New operation to perform. DoNew is called when the user selects "New" from the File menu.

```
virtual BOOLEAN DoOpen(void);
```

Creates a shell document and then gives it an Open operation to perform. DoOpen is called when the user selects "Open" from the File menu.

```
virtual void SetUpMenus(void);
```

A method called to initialize the menubar. It enables the "New" and "Open" items on the File menu so that they can be activated.

Methods Inherited From CApplication But Not Overridden

Methods are listed alphabetically by name.

```
virtual void AddDocument(CDocument* theDocument);
virtual void ChangeFont(FONT theFont, FONT_PART thePart);
virtual void CloseAll(void);
virtual CDocument* FindDocument(int theId) const;
virtual void DoAboutBox(void);
virtual BOOLEAN DoClose(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual BOOLEAN DoPageSetUp(void);
virtual BOOLEAN DoPrint(void);
virtual BOOLEAN DoSave(void);
virtual BOOLEAN DoSaveAs(void);
virtual BOOLEAN DoQuit(void);
virtual const CEnvironment* GetEnvironment(void) const;
int GetNumDocuments(void);
virtual const CList* GetSubObjects(void) const = NULL;
virtual void RemoveDocument(CDocument* theDocument);
```

```
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN  
isUpdate = FALSE);  
virtual void UpdateMenus(void);
```

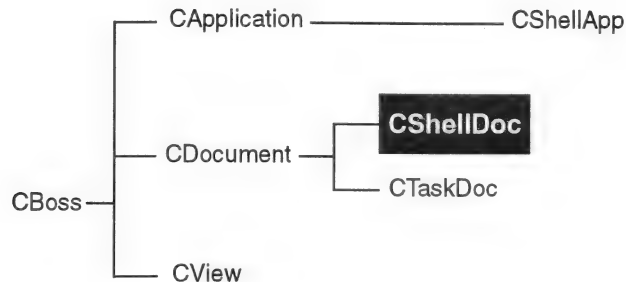
Overrides

In most cases, you should add methods to this class rather than override its methods.

Summary: CShellApp.h

```
#ifndef CShellApp_H  
#define CShellApp_H  
  
#ifdef STDH  
#include "CApplication.h"  
#endif STDH  
#ifdef DOS  
#include "CApplctn.h"  
#endif DOS  
  
class CShellApp : public CApplication  
{  
public:  
    CShellApp(void);  
  
    virtual void StartUp(void);  
    virtual void DoCommand(long theCommand, void* theData=NULL);  
    virtual BOOLEAN DoNew(void);  
    virtual BOOLEAN DoOpen(void);  
    virtual void SetUpMenus(void);  
};  
#endif CShellApp_H
```

CShellDoc



Description

CShellDoc is an example of how to override the minimal set of methods of CDocument. It initializes the inherited application and creates a shell window.

Heritage

Superclass: CDocument

Usage

Rename this class and modify it as necessary—or simply use it as an example.

Data Members

None.

Public Methods

```
CShellDoc(CApplication *theApplication, int theId);
```

Takes a document ID and calls that document's constructor.

```
virtual void BuildWindow(void);
```

Creates a new shell window and initializes that window with a title.

```
virtual void DoCommand(long theCommand, void* theData=NULL);
```

An example of how a CDocument-derived class can override the DoCommand method. The void pointer theData can be used to pass any user-defined structure to the DoCommand method.

virtual BOOLEAN DoNew(void);

This method, called when the user selects “New” from the File menu, calls BuildWindow to create a new document window.

virtual BOOLEAN DoOpen(void);

This method is called when the user selects “Open” from the File menu. It calls the inherited DoOpen and calls BuildWindow.

Methods Inherited From CDocument But Not Overridden

Methods are listed alphabetically by name.

virtual void AddWindow(CWindow * theWindow);

virtual void ChangeFont(FONT theFont, FONT_PART thePart);

virtual BOOLEAN DoClose(void);

virtual void CloseAll(void);

virtual void DoKey(int theKey, BOOLEAN theShiftKey, BOOLEAN theControlKey);

virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);

virtual BOOLEAN DoPageSetUp(void);

virtual BOOLEAN DoPrint(void);

virtual BOOLEAN DoSave(void);

virtual BOOLEAN DoSaveAs(void);

virtual const CEnvironment* GetEnvironment(void) const;

virtual CWindow* FindWindow(int theId) const;

int GetId(void) const;

int GetNumWindows(void);

virtual const CList* GetSubObjects(void) const = NULL;

virtual void SetId(int theId);

BOOLEAN GetSave(void) const;

virtual void RemoveWindow(CWindow *theWindow);

virtual void SetEnvironment(const CEnvironment& theEnvironment);

void SetSave(BOOLEAN isSaved);

virtual void UpdateMenus(void);

Overrides

In most cases, you should add methods to this class rather than overriding its methods.

Summary: CShellDoc.h

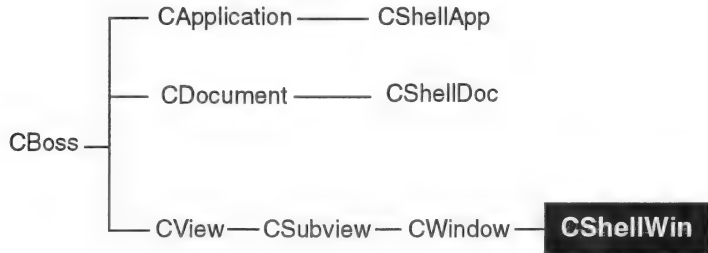
```
#ifndef CShellDoc_HXX
#define CShellDoc_HXX

#ifdef STDH
#include "CDocument.h"
#endif STDH
#ifdef DOS
#include "CDocmnt.h"
#endif DOS

class CShellDoc : public CDocument
{
public:
    CShellDoc(CApplication *theApplication,int theId );
    virtual void BuildWindow(void);
    virtual void DoCommand(long theCommand,void* theData=NULL);
    virtual BOOLEAN DoNew(void);
    virtual BOOLEAN DoOpen(void);
};

#endif CShellDoc_HXX
```

CShellWin



Description

CShellWin is an example of how to override the minimal set of methods of CWindow. It initializes the inherited application and creates a shell document.

Heritage

Superclass: CWindow

Usage

Rename this class and modify it as necessary—or simply use it as an example.

Environment

The properties of the the window's border are system-defined. If the window's background it set to TRUE, its interior is painted with the brush, which has settings for color and pattern.

Data Members

None.

Public Methods

```
CShellWin(CDocument *theDocument, const CRect&
theRegion, BOOLEAN isClosable, BOOLEAN
isSizable, WIN_TYPE theWindowType);
```

The constructor. It is an example of how CWindow-derived classes can be initialized while initializing their CWindow-inherited data. Here is where you would add code to create the views the window should contain.

```

    BOOLEAN IShellWin(BOOLEAN isBackgroundDrawn= TRUE,
                      const CString theTitle    = NULLString,
                      BOOLEAN isVisible         = TRUE);

```

The initializer. It is an example of how to create an initializer for a CWindow-derived class. Add code here to initialize your window class.

```

virtual void DoCommand(long theCommand, void*
                      theData = NULL);

```

This method always calls the inherited as the default, that is, the DoCommand of CWindow. The void pointer theData can be used to pass any user-defined structure to the DoCommand method.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```

virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);

```

From CView

```

virtual void Activate(void);
virtual void Deactivate(void);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;

```

```

virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);

```



```

virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation,List* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);

```

From CWindow

```
virtual void ChangeFont(FONT theNewFont, FONT_PART thePart);
virtual void Close(void);
virtual void Disable(void);
virtual void DoActivateWindow(void);
virtual void DoDeactivateWindow(void);
void DoHScroll(SCROLL_CONTROL theEvent, short thePos);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
void DoVScroll(SCROLL_CONTROL theEvent, short thePos);
virtual void Draw(const CRect& theClippingRegion);
virtual void Enable(void);
virtual CRect GetClippedFrame(void) const;
virtual CDocument* GetDocument(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual BOOLEAN IsBackgroundDrawn(void) const;
virtual BOOLEAN IsClosable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void SetBackgroundDrawing(BOOLEAN isBackgroundDrawn);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetSelectedView(CView *theSubview);
virtual void SetTitle(const CString& theNewTitle);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void SizeWindow(int theWidth, int theHeight);
virtual void UpdateMenus(void);
virtual void UpdateUnits(CUnits* theUnits);
```

Overrides

In most cases, you should add methods to this class rather than overriding its methods.

Summary: CShellWin.h

```
#ifndef CShellWin_HXX
#define CShellWin_HXX

#include "CWindow.h"

class CShellWin : public CWindow
{
public:
    CShellWin(CDocument *theDocument,
              const CRect& theRegion,
              BOOLEAN isClosable,
              BOOLEAN isSizable,
              WIN_TYPE theWindowType);

    BOOLEAN IShellWin(BOOLEAN isBackgroundDrawn = TRUE,
                      const CString theTitle = NULLString,
                      BOOLEAN isVisible = TRUE);
    virtual void DoCommand(long theCommand, void* theData = NULL);
};

#endif CShellWin_HXX
```

CSketchPad

CView — CSubview — **CSketchPad**

Description

CSketchPad is a class that allows you to designate an area on the computer screen as a sketching area. A CSketchPad object is an area inside a CView that makes it easy to “draw” other subviews. The user can drag the mouse within this area, press the mouse button, and move the mouse to drag out a certain shape—a rectangle or line. This class is useful for building drawing programs.

Heritage

Superclass: CSubview

Usage

To drag areas inside the sketchpad, the user presses and holds down the mouse button while dragging the mouse. Upon receiving a MouseUp event, the sketchpad generates a DoCommand message. Usually you should call the GetSketchedRegion or GetSketchPoint methods upon receiving the sketchpad’s DoCommand message (these methods return valid values only during DoCommand message handling).

Use the sketchpad as an enclosure for all other views in the sketchable region. You can call the SetSketchEverywhere method either to restrict sketching to the empty area of the sketchpad or to enable sketching over the views nested inside as well as over the empty area.

Environment

The border of the sketchpad is drawn with the *pen*, and its interior is painted with the *brush*. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

CSketchFrame*	itsSketchFrame;	the wire frame to draw
BOOLEAN	itIsReady;	whether there is an active wire frame
SKETCHTYPE	itsSketchType;	type of sketch drawn
BOOLEAN	itIsSketchEverywhere;	sketch over nested views

Public Methods

Constructor, Destructor, and Initializer Methods

CSketchPad(CSubview*theEnclosure, const CRect& theRegion);

A constructor. theEnclosure is a pointer to the subview that will contain the sketchpad. theRegion is a coordinate location, local to the enclosure, that is used to place the sketchpad.

CSketchPad(const CSketchPad& theSketchPad);

A copy constructor that duplicates the attributes, but not the contents, of a sketchpad. That is, it copies the grid but does not do a deep copy of the objects it contains.

CSketchPad& operator=(const CSketchPad& theSketchPad);

An assignment operator that duplicates the attributes, but not the contents, of a sketchpad. That is, it copies the grid but does not do a deep copy of the objects it contains.

~CSketchPad(void);

The destructor. It cleans up after the sketchpad and deletes any views nested within it.

**BOOLEAN ISketchPad(SKETCHTYPE theType= RECTANGLE,
BOOLEAN isVisible = TRUE,
long theGlue = NULLSTICKY);**

The initializer. To initialize the sketchpad, first give it a sketch type. There are two possible values for this parameter: RECTANGLE or LINE, which designate what kind of sketching is desired. In addition to the sketch type, this initializer takes a visibility state and a glue type. One or more of its defaults can be overridden.

Sketch Event Methods

When you drag out a shape on the sketchpad, as soon as you release the mouse button, the shape disappears and a DoCommand is generated. Your application should react to the command by calling one of the three sketch event methods described here. Note that these methods can only be called as a response to a CSketchPad DoCommand. Calling them at any other time results in inaccurate return values. All of these methods return coordinates that are relative to the sketchpad.

virtual CRect GetSketchedRegion() const;

Returns a region—that is, a CRect—in coordinates that are relative to the sketchpad. These coordinates indicate the size of the region that has been dragged out. If you drag out a rectangular region, the rectangle is returned. If you drag out a line, a rectangular region that fits the whole line is returned.

virtual CPoint GetStartPoint(void) const;

Returns the coordinate for the point at which mouse dragging started. This coordinate is relative to the sketchpad.

virtual CPoint GetEndPoint() const;

Returns the coordinate for the point at which mouse dragging ended. This coordinate is relative to the sketchpad.

virtual void SetSketchType(SKETCHTYPE theType = RECTANGLE);

Sets the sketch type either to a value of LINE or of RECTANGLE, designating the type of sketching desired.

virtual SKETCHTYPE GetSketchType(void) const;

Returns the current sketch type, which is a value of either LINE or RECTANGLE.

virtual void SetSketchEverywhere(BOOLEAN isSketchEverywhere);

Allows you to set whether mouse events are received by the sketchpad (TRUE) or by an object inside the sketchpad (FALSE). When isSketchEverywhere is set to TRUE, the user can draw overlapping objects and even sketch one object on top of another. When it is set to FALSE, sketching can occur only over the empty space within the sketchpad.

virtual BOOLEAN IsSketchEverywhere(void);

Returns a Boolean value indicating whether isSketchEverywhere is in effect (TRUE) or not (FALSE). If it is

in effect, the user can draw objects over other objects as well as over the empty space in the sketchpad.

Inherited Utilities

CSketchPad overrides two mouse methods to implement the dragging out of a shape. Mouse events are passed directly to the view at the location where the event occurs. Each of the following methods has a `theButton` argument that specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right). By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button.

```
virtual voidMouseDown(CPoint theLocation,
                      short theButton    = 0,
                      BOOLEAN isShiftKey  = FALSE,
                      BOOLEAN isControlKey= FALSE);
```

When the user presses down a mouse button over a sketchpad containing local coordinate `theLocation`, the sketchpad receives and handles this event.

```
virtual voidMouseUp(CPoint theLocation,
                    short theButton    = 0,
                    BOOLEAN isShiftKey  = FALSE,
                    BOOLEAN isControlKey= FALSE);
```

When the user releases a mouse button over a sketchpad containing local coordinate `theLocation`, the sketchpad receives and handles this event. A sketch command is generated.

```
virtual CView* FindEventTarget(const CPoint&
                               theLocation) const;
```

A method that is called to find the target of a mouse event within a subview. This virtual method is defined to return the deepest subview at the global coordinates of `theLocation`. This method can be overridden by a subview in order to trap all events regardless of the subviews it contains or to change the event targeting algorithm. For example, a sketchpad can contain many subviews (graphical shapes). When you click on an object inside a sketchpad, the event does not go to that object. Instead, the sketchpad overrides `FindEventTarget` and traps the event by returning this (itself) as the target for the mouse events received. Then, it treats the mouse events as necessary, without sending them on to its subviews. However, when the sketchpad is disabled or when `SetSketchEverywhere` is turned off, then the event does go to the object inside the sketchpad rather than to the sketchpad itself:

```
virtual void Draw(const CRect&
theClippingRegion);
```

A method that has been overridden so a user can draw the sketchpad. Sketchpads look like rectangles with a border drawn in the pen color and an interior painted in the brush color.

```
virtual void Enable(void);
```

Enables the sketchpad to receive events.

```
virtual void Disable(void);
```

You do not always want the sketchpad to catch all of the events; sometimes you want the events to go straight down to the views themselves or to another part of the window. Thus, `Disable` disables the sketchpad so that it cannot receive events and the user cannot sketch in it.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

```
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
```

```
virtual void Deactivate(void);
```

```
virtual void DoCommand(long theCommand, void* theData=NULL);
```

```
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
```

```
virtual void DoTimer(long theTimerId);
```

```
virtual void DoUser(long theUserId, void* theData);
```

```
virtual void Draw(void);
```

```
virtual CView* FindHitView(const CPoint& theLocation) const;
```

```
virtual CRect GetClippedFrame(void) const;
```

```
virtual long GetCommand(void) const;
```

```
virtual CWindow* GetCWindow(void) const;
```

```
virtual long GetDoubleCommand(void) const;
```



```
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
```

```

virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;

```

```

virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

Summary: CSketchPad.h

```

#ifndef SketchPad_H
#define SketchPad_H
#include "CSubview.h"

typedef enum
{
    RECTANGLE, LINE
}
SKETCHTYPE;
class CSketchFrame;
class CSketchPad : public CSubview
{
public:
    CSketchPad(CSubview *theEnclosure, const CRect& theRegion);
    CSketchPad(const CSketchPad& theSketchPad);
    CSketchPad& operator=(const CSketchPad& theSketchPad);
    ~CSketchPad(void);

    BOOLEAN ISketchPad(SKETCHTYPE theType      = RECTANGLE,
                      BOOLEAN isVisible         = TRUE,
                      long theGlue             = NULLSTICKY);

    // SketchPad utility:
    virtual CRect GetSketchedRegion() const;

    virtual CPoint GetStartPoint(void) const;
    virtual CPoint GetEndPoint() const;

    virtual void SetSketchType(SKETCHTYPE theType = RECTANGLE);
    virtual SKETCHTYPE GetSketchType(void) const;

    virtual void SetSketchEverywhere(BOOLEAN isSketchEverywhere);
    virtual BOOLEAN IsSketchEverywhere(void);

    // Inherited utility:
    virtual void MouseDown(CPoint theLocation, short theButton = 0,
                          BOOLEAN isShiftKey=FALSE,
                          BOOLEAN isControlKey=FALSE);

```

```
        virtual void MouseUp(CPoint theLocation, short theButton = 0,
                               BOOLEAN isShiftKey=FALSE,
                               BOOLEAN isControlKey=FALSE);

    virtual CView* FindEventTarget(const CPoint& theLocation) const;

    virtual void Draw(const CRect& theClipping);

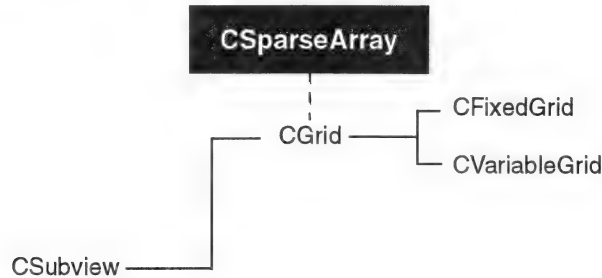
        virtual void Enable(void);
        virtual void Disable(void);

    private:

        CSketchFrame* itsSketchFrame; // the wire frame to draw
        BOOLEAN itIsReady;           // is there an active wire frame
        SKETCHTYPE itsSketchType;    // type of sketch drawn
};

#endif SketchPad_H
```

CSparseArray



Description

CSparseArray is a container class that stores data in a two-dimensional array. This class is useful when the data stored is sparse because the number of array cells (memory) is equal to the number of nonempty array locations. Thus, **CSparseArray** conserves memory, allocating memory only when it is needed. **CSparseArray** is an XVT-Power++ internal class; if you want to use it, keep in mind that it is not fully supported in this release of XVT-Power++.

Heritage

Superclass: none

Usage

Instantiate a **CSparseArray** object and operate on it using the interface methods described here. The **CSparseArrayIterator** is provided to iterate efficiently through the items in the sparse array.

Because of the use of `void*` you must use casting in at least two places when using **CSparseArray** under the current non-template implementation. Here is an example of use:

```

CSparseArray aSparseArray();
Foo* aFoo = new Foo(1,2,3);
aSparseArray.Insert((void*) aFoo, 3, 8); // Cast #1
...
Foo* fooItem;
CSparseArrayIterator doTo(aSparseArray);
while (fooItem = (Foo*) doTo.Next()) // Cast #2
    fooItem->Goo();
  
```

Friends

```
friend class CSparseArrayIterator;
friend class CSparseRowIterator;
friend class CSparseColIterator;
```

Public Methods

Constructor and Destructor Methods

```
CSparseArray(void);
```

A constructor, which creates an empty array.

```
CSparseArray(const CSparseArray&
theSparseArray);
```

A copy constructor, which copies all of the attributes of theSparseArray, including any pointers it contains. It does not copy the objects to which the array is pointing.

```
CSparseArray& operator=(const CSparseArray&
theSparseArray);
```

An assignment operator, which copies all of the attributes of theSparseArray, including any pointers it contains. It does not copy the objects to which the array is pointing.

```
virtual ~CSparseArray(void);
```

The destructor, which cleans up after the array but does not delete the objects to which it is pointing.

Operations

```
virtual void Insert(void *item, int row, int
col);
```

Inserts an item into an array, at the position of the given row and column. This method allows multiple items to be inserted at a position. item is a void pointer to a generic object.

```
virtual BOOLEAN Remove(void *item);
```

Removes an item from the array. That is, it removes a void pointer (item) to a generic object. It returns a Boolean value of TRUE if it succeeds. If an item is multiply stored in the array, this method removes only the first occurrence of this item that it encounters.

```
virtual void* Remove(int row, int col);
```

Removes the item from the position specified by the given row and column. It removes a void pointer (item) to a generic object. The method returns a pointer to the removed item or NULL if no item is found.

```
virtual void* GetContents(int row, int col)
    const;
```

Returns a pointer to the item located at the position specified by the given row and column. It returns NULL if there is no item at this position.

```
virtual int GetNumRows(void) const;
```

Returns the number of rows in the array.

```
virtual int GetNumCols(void) const;
```

Returns the number of columns in an array.

Iterators for CSparseArray CSparseArrayIterator

Description

CSparseArrayIterator iterates through all of the cells of an array in no guaranteed order.

Usage

See CIterator.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

int	itsRowMarker;	
int	itsColMarker;	
COrderedList	*itsRows;	
COrderedList	*itsCurrentColList;	
int	itsMaxRows;	the maximum number of rows over which to iterate
int	itsMaxCols;	the maximum number of columns over which to iterate

Public Methods

**CSparseArrayIterator(const CSparseArray
*aSparseArray);**

A constructor. It takes a pointer to a sparse array,
aSparseArray.

**CSparseArrayIterator(const
CSparseArrayIterator& theIterator);**

A copy constructor, which duplicates all of the attributes of a
sparse array iterator.

**CSparseArrayIterator& operator=(const
CSparseArrayIterator& theIterator);**

An assignment operator, which duplicates all of the attributes of
a sparse array iterator..

~CSparseArrayIterator(void);

The destructor.

void* Next(int *row=NULL,int *col=NULL);

Returns the item in the position specified by the row and
column numbers each time it is called. When it reaches the end
of the list, it returns NULL. You can iterate by starting a loop and
continuing the loop as long as NEXT returns nonNULL values.

Iterators for SparseArray CSparseRowIterator

CSparseRowIterator iterates through all of the cells in a specified
row of an array.

Private Data Members

COorderedIterator* itsIterator;

Public Methods

**CSparseRowIterator(const CSparseArray*
theSparseArray, int theRow);**

A constructor. It takes an array over which to iterate and the
number of the row through which it is to iterate.

**CSparseRowIterator(const CSparseRowIterator&
theIterator);**

A copy constructor. It duplicates all of the attributes of a sparse
row iterator.


```
CSparseRowIterator& operator=(const
    CSparseRowIterator& theIterator);
```

An assignment operator, which duplicates all of the attributes of a sparse row iterator.

```
~CSparseRowIterator(void);
```

The destructor.

```
void* Next(int* theCol = NULL);
```

Returns the next item in the array. If there is no item, it returns NULL.

Iterators for SparseArray CSparseColIterator

CSparseColIterator iterates through all of the cells in a given column of an array.

Private Data Members

COderedIterator	itsRowIterator;
int	itsCurrentCol;

Public Methods

```
CSparseColIterator(const CSparseArray*
    theSparseArray, int theCol);
```

A constructor. It takes an array over which to iterate and the number of the column through which it is to iterate.

```
CSparseColIterator(const CSparseColIterator&
    theIterator);
```

A copy constructor. It duplicates all of the attributes of a column iterator.

```
CSparseColIterator& operator=(const
    CSparseColIterator& theIterator);
```

An assignment operator, which duplicates all of the attributes of a column iterator.

```
~CSparseColIterator(void);
```

The destructor.

```
void* Next(int* theRow = NULL);
```

Returns the item in the position specified by the row number each time it is called. When it reaches the end of the list, it returns NULL. You can iterate by starting a loop and continuing the loop as long as NEXT returns nonNULL values.

Summary: CSparseArray.h

```

#ifndef CSparseArray_H
#define CSparseArray_H

#include "PwrNames.h"
#include "xvt.h"

#include COrderedList_i

class CSparseArray
{
    friend class CSparseArrayIterator;
    friend class CSparseRowIterator;
    friend class CSparseColIterator;

public:
    CSparseArray(void);
    CSparseArray(const CSparseArray& theSparseArray);
    CSparseArray& operator=(const CSparseArray& theSparseArray);
    virtual ~CSparseArray(void);

    // Operations:
    virtual void Insert(void *item, int row, int col);
    virtual BOOLEAN Remove(void *item);
    virtual void* Remove(int row, int col);
    virtual void* GetContents(int row, int col) const;

    virtual int GetNumRows(void) const;
    virtual int GetNumCols(void) const;

private:
    COrderedList *itsRows;
    int itsRowNumber;
    int itsColNumber;
};

class CSparseArrayIterator
{
public:
    CSparseArrayIterator(const CSparseArray *aSparseArray);
    CSparseArrayIterator(const CSparseArrayIterator& theIterator);
    CSparseArrayIterator& operator=(const CSparseArrayIterator&
                                   theIterator);
    ~CSparseArrayIterator(void);

    void* Next(int *row=NULL, int *col=NULL);

private:
    int itsRowMarker;
    int itsColMarker;
    COrderedList *itsRows;
    COrderedList *itsCurrentColList;
    int itsMaxRows;
    int itsMaxCols;
};

class CSparseRowIterator
{
public:

```

```
    CSparseRowIterator(const CSparseArray* theSparseArray, int theRow);
    CSparseRowIterator(const CSparseRowIterator& theIterator);
    CSparseRowIterator& operator=(const CSparseRowIterator& theIterator);
    ~CSparseRowIterator(void);

    void* Next(int* theCol = NULL);

private:
    COrderedIterator* itsIterator;
};

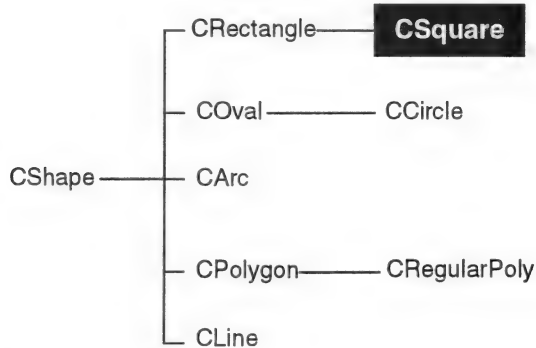
class CSparseColIterator
{
public:
    CSparseColIterator(const CSparseArray* theSparseArray, int theCol);
    CSparseColIterator(const CSparseColIterator& theIterator);
    CSparseColIterator& operator=(const CSparseColIterator& theIterator);
    ~CSparseColIterator(void);

    void* Next(int* theRow = NULL);

private:
    COrderedIterator itsRowIterator;
    int itsCurrentCol;
};

#endif CSparseArray_H
```

CSquare



Description

CSquare objects paint a square inside a CSubview.

Heritage

Superclass: CRectangle

Usage

Create a CSquare object and initialize it. Like all shapes, this class inherits all properties of CSubview objects, such as stickiness, enclosure, and so on.

Environment

The lines of the square are drawn with the *pen*; its interior is painted with the *brush*. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width.

Data Members

None.

Public Methods

```
CSquare(CSubview* theEnclosure, const CPoint& theTopLeftPoint, int theSideLength);
```

A constructor. theEnclosure is a pointer to the subview that will contain the square. Also, this constructor requires the origin (theTopLeftPoint) and the length for the square's sides (theSideLength).

CSquare(const CSquare& theSquare);

A copy constructor that creates a new CSquare object with the same enclosure, color, visibility attributes, enabled/disabled attributes, environment, and so on as the original CSquare object. However, any views nested within the original CSquare object are not copied.

CSquare& operator=(const CSquare& theSquare);

An assignment operator. It copies the attributes of the original CSquare object, creating a new CSquare object that has the same color, glue, environment, visibility state, and so on. However, any shapes nested within the original CSquare object are not copied.

```
BOOLEAN ISquare(BOOLEAN hasRoundCorners= FALSE,  
                int theCornerWidth      = 0,  
                int theCornerHeight    = 0,  
                BOOLEAN isVisible      = TRUE,  
                GLUETYPE theGlue      = NULLSTICKY);
```

The initializer. Squares can optionally have rounded corners. If this is desired, set the Boolean parameter `hasRoundCorners` to `TRUE`. If this parameter is set to `TRUE`, then you must also set the `theCornerWidth` and `theCornerHeight` parameters, which take numbers indicating how high and how deep the rounding be. This initializer also allows the visibility status and the glue type of the square to be set.

virtual void Size(const CRect& theNewSize);

Sizes the square according to the coordinates of `theNewSize`. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, `theNewSize`, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated. If the width of `theNewSize` is not equal to the height, `Size` takes the average to calculate the square's new size.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);  
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);  
virtual void Deactivate(void);  
virtual void Disable(void);  
virtual void DoCommand(long theCommand,void* theData=NULL);  
virtual BOOLEAN DoPrint(const CRect& theRegion) const;  
virtual void DoTimer(long theTimerId);  
virtual void DoUser(long theUserId, void* theData);  
virtual void Draw(void);  
virtual void Enable(void);  
virtual CView* FindHitView(const CPoint& theLocation) const;  
virtual CRect GetClippedFrame(void) const;  
virtual long GetCommand(void) const;  
virtual CWindow* GetCWindow(void) const;  
virtual long GetDoubleCommand(void) const;  
virtual CSubview* GetEnclosure(void);  
virtual const CEnvironment* GetEnvironment(void) const;  
virtual CRect GetFrame(void) const;  
virtual CRect GetGlobalFrame(void) const;  
virtual CPoint GetGlobalOrigin(void) const;  
virtual GLUETYPE GetGlue(void) const;  
virtual int GetId(void) const;  
virtual CRect GetLocalFrame(void) const;  
virtual CPoint GetOrigin(void) const;  
virtual const CString GetTitle(void) const;  
virtual CWireFrame* GetWireFrame(void) const;  
virtual void Glue();  
virtual void Hide(void);  
virtual void HScroll(SCROLL_CONTROL theType, int thePos);  
virtual BOOLEAN IsActive(void) const;  
virtual BOOLEAN IsEnabled(void) const;  
virtual BOOLEAN IsEnvironmentShared(void);  
virtual BOOLEAN IsDraggable(void) const;
```

```

virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);

```

```

virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation,CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjeects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CShape

None.

From CRectangle

```
virtual void Draw(const CRect& theClippingRegion);
```

```
virtual void SetRoundedCorners(BOOLEAN isCornerRounded, int
theCornerWidth=SAME,int theCornerHeight=SAME);
```

Summary: CSquare.h

```
#ifndef CSquare_H
#define CSquare_H

#ifdef STDH
#include "CRectangle.h"
#endif
#ifdef DOS
#include "CRectngl.h"
#endif

class CSquare : public CRectangle
{
public:
    CSquare(CSubview* theEnclosure,
            const CPoint& theTopLeftPoint, UNITS theSideLength);
    CSquare(const CSquare& theSquare);
    CSquare& operator=(const CSquare& theSquare);

    BOOLEAN ISqaure(BOOLEAN hasRoundCorners      = FALSE,
                    UNITS theCornerWidth          = 0,
                    UNITS theCornerHeight         = 0,
                    BOOLEAN isVisible             = TRUE,
                    GLUETYPE theGlue              = NULLSTICKY);

    virtual void Size(const CRect& theNewSize);
};

#endif CSquare_H
```

CStartup

CMem	CGlue	CSwitchBoard
CError	CDesktop	CStartup
	CEnvironment	CGlobalClassLib
		CGlobalUser
		CResourceMgr

Description

CStartup provides the place where your program transfers control to XVT-Power++ inside of main. Within the **CStartUp.cxx** file, you create an instance of your user-derived CApplication class and invoke a Go method. An example of this procedure is shown here:

```
void main(int argc, char* argv[])
{
    CMyApp theApplication;
    theApplication.Go(argc, argv, MY_MENU_BAR_RID, ABOUTBOX,
        "BaseName", "Application Name",
        "Task Title");
}
```

No code should follow the call to Go since XVT-Power++ never returns control to your program once the Go message is processed. In addition, no XVT-Power++ graphical objects can be created before the call to Go. Your application object receives a Startup event when the system is ready to start executing your application.

CString

CString

CPoint

CRect

CUnits

Description

CString is a class representation of character strings. Its purpose is to encapsulate the data structures and utilities of character strings, making their usage much simpler.

The CString class encapsulates the notion of a storage size for a string. The length of a string is determined by the number of non-NULL characters. The string may have room to store many more characters, or it may be exactly the size to accommodate the characters it contains. The storage capacity of a string is completely hidden from the user and has several implications. "Hidden" means that if a user concatenates two strings or uses an indexing function within a string, CString automatically takes care of expanding or shrinking the string's allocated memory for the user.

There are two exceptions, however, that allow the programmer to get access to the internal string storage. They are the conversion operator that converts strings to char pointers and the indexing operator that returns a char reference. Both of these methods could certainly have been made completely safe so that they do not return references or pointers to internal structures. We had to make a choice between an interface to CString that is safe but cumbersome and an interface that is clean, convenient, and almost safe but contains exceptions that the user must treat cautiously. We chose the latter alternative. Because strings are used in numerous places throughout the XVT-Power++ code, the string class must be very easy, simple and efficient to use. Once you have used the string class for awhile and understand how it works, you should have no problems with the two exceptions.

Another important feature of CString is that it maintains reference counting. For the sake of efficiency when strings are copied, passed

by value, and so on, we don't actually allocate new memory. Rather, we store references to the places in memory and keep a count. As long as different reference copies are not modified, we're safe. As soon as one reference copy is modified and there is more than one reference to that string, then we do the allocation, just as we do with lists. Strings reap the same benefits from reference counting as lists (see CList).

Heritage

Superclass: none

Usage

You can construct strings from a char pointer, a string resource ID, or another CString object. Several operators have been defined to manipulate strings. All string storage growth and shrinkage is handled internally, and you need not worry about size.

Be sure to read this section carefully before using the CString class.

Data Members

None.

Private Data Members

CStringBody	*itsBody;	an internal string representation class
static unsigned int	BUFSIZE;	
static char	itsDelimiter;	
static BOOLEAN	itConsumesDelimiter	

Public Methods

Class Utility

CString(void);

The default constructor, which creates an empty string.

CString(const char* theString);

A constructor, which takes a constant char pointer. An internal copy of theString is made.

CString(char theDisownedString);**

A constructor that takes a pointer to a string pointer. An internal deep copy is not made, but rather a shallow copy is made internally. In addition, this method sets the value of the pointer

that is passed in to NULL, making it unusable after a call to this constructor is made.

CString(const CString& theString);

A copy constructor, which copies one string into the other.

CString(int theResourceId, int theStringSize = 64);

A constructor. It takes a resource ID number, which is the ID of a string resource that is defined in the URL file. It also takes a string size (in number of characters) for that resource ID. When CString reads a string from the resources, it needs to know how many characters to read for that string ID. By default, this method seeks to read 64 characters. If there are fewer than 64 characters, there is no problem. If there are more than 64 characters, then the user must specify the size of the string so that this resource can be read in. Inside the URL file, string resources are numbered in *descending* consecutive order. For portability reasons, string resources in the URL file cannot be numbered in an ascending consecutive order. If they are, problems will result. If you are defining resources for the Macintosh, for instance, a string list resource will be created instead of the usual string resource, and this is not what is desired.

CString& operator=(const CString& theString);

An assignment operator, which allows you to set one string equal to another.

~CString(void);

The destructor.

Conversion Operators

When you use the conversion operator, be sure not to store, delete, or reallocate the pointer returned. If you need a non-const char*, you can use GetCharPtr, but do not store it or modify its return value.

operator const char*(void) const;

Returns a constant pointer to a char string. By converting a CString to a constant char pointer, you can use it wherever a char pointer is needed. Of course, you cannot delete what the pointer is pointing to, and you cannot cast away constants and change its contents.

The pointer returned may become invalid the next time the string object is modified. Therefore, do not store the value of the

pointer returned and expect to use it safely throughout your program.

```
char* GetCharPtr(void) const;
```

Returns a pointer that must be treated as a constant. All restrictions that apply to the conversion operator apply to this method as well. Here you do not get the protection of a constant and you should use it cautiously.

Concatenating and Appending

```
CString& operator+=(const CString&  
theStringToCatenate);
```

Defines the plus-equal operator for concatenating strings.

```
CString& operator+=(const char*  
theStringToCatenate);
```

Defines the plus-equal operator for concatenating char pointers.

```
friend CString operator+(const CString&  
theLeftString, const char* theRightString);
```

Appends a char pointer (theRightString) to a string (theLeftString).

```
friend CString operator+(const char*  
theLeftString, const CString&  
theRightString);
```

Appends a string (theRightString) to a char pointer (theLeftString).

```
friend CString operator+(const CString&  
theLeftString, const CString&  
theRightString);
```

Appends theRightString to theLeftString and returns a temporary string object with the result.

```
void Append(const CString& theString, int  
theNumberOfCharacters);
```

A method that is similar to the methods for concatenating, except that it allows you to pass in a string and also to specify the number of characters to be appended. This method is useful if you do not want to append the entire string.

```
void Append(const char* theString, int  
theNumberOfCharacters);
```

A method that is similar to the previous methods for concatenating, except that it allows you to pass in a char pointer

and to specify the number of characters to be appended. This method is useful if you do not want to append the entire string.

Comparison

The following comparison operators are multiply overloaded to avoid ambiguity. The comparisons are case-sensitive and follow alphanumeric comparison order.

Equality Comparisons:

The following three methods are equality operators that return a Boolean value of TRUE or FALSE, depending on whether the strings compared are lexically the same.

```
friend BOOLEAN operator==(const CString&
    theLeftString, const CString&
    theRightString);
```

Compares two strings, theLeftString and theRightString.

```
friend BOOLEAN operator==(const CString&
    theLeftString, const char* theRightString);
```

Compares a string (theLeftString) with a char pointer (theRightString).

```
friend BOOLEAN operator==(const char*
    theLeftString, const CString&
    theRightString);
```

Compares a char pointer (theLeftString) with a string (theRightString).

Inequality Comparisons:

The following three methods are not-equal operators, which return a Boolean value of TRUE if the strings are lexically not equal.

```
friend BOOLEAN operator!=(const CString&
    theLeftString, const CString&
    theRightString);
```

Compares two strings, theLeftString and theRightString.

```
friend BOOLEAN operator!=(const CString&
    theLeftString, const char* theRightString);
```

Compares a string (theLeftString) with a char pointer (theRightString).

```
friend BOOLEAN operator!=(const char*
    theLeftString, const CString&
    theRightString);
```

Compares a char pointer (theLeftString) with a string (theRightString).

Greater-Than Comparisons

The following three methods are greater-than operators. “Greater-than” means that the first letter of one string is higher in the alphabet than the first letter of the other. *B* is higher in the alphabet than *A*.

```
friend BOOLEAN operator>(const CString&
    theLeftString, const CString&
    theRightString);
```

Compares two strings, theLeftString and theRightString. This method returns a Boolean value of TRUE if theLeftString is greater-than theRightString.

```
friend BOOLEAN operator>(const char*
    theLeftString, const CString&
    theRightString);
```

Compares a char pointer (theLeftString) with a string (theRightString). It returns a Boolean value of TRUE if the char pointer is greater-than the string.

```
friend BOOLEAN operator>(const CString&
    theLeftString, const char* theRightString);
```

Compares a string (theLeftString) with a char pointer (theRightString). It returns a Boolean value of TRUE if the string is greater-than the char pointer.

Less-Than Comparisons

The following three methods are less-than operators. “Less-than” means that the first letter of one string is lower in the alphabet than the first letter of the other. *A* is lower in the alphabet than *B*.

```
friend BOOLEAN operator<(const CString&
    theLeftString, const CString&
    theRightString);
```

Compares two strings, theLeftString and theRightString. This method returns a Boolean value of TRUE if theLeftString is less-than theRightString.


```
friend BOOLEAN operator<(const char*  
    theLeftString, const CString&  
    theRightString);
```

Compares a char pointer (theLeftString) with a string (theRightString). It returns a Boolean value of TRUE if the char pointer is less-than the string.

```
friend BOOLEAN operator<(const CString&  
    theLeftString, const char* theRightString);
```

Compares a string (theLeftString) with a char pointer (theRightString). It returns a Boolean value of TRUE if the string is less-than the char pointer.

Greater-Than-Or-Equal Comparisons

The following three methods are greater-than-or-equal operators. “Greater-than” means that the first letter of one string is higher in the alphabet than the first letter of the other. *B* is higher in the alphabet than *A*.

```
friend BOOLEAN operator>=(const CString&  
    theLeftString, const CString&  
    theRightString);
```

Compares two strings, theLeftString and theRightString. This method returns a Boolean value of TRUE if theLeftString is greater-than-or-equal to theRightString.

```
friend BOOLEAN operator>=(const char*  
    theLeftString, const CString&  
    theRightString);
```

Compares a char pointer (theLeftString) with a string (theRightString). It returns a Boolean value of TRUE if the char pointer is greater-than-or-equal to the string.

```
friend BOOLEAN operator>=(const CString&  
    theLeftString, const char* theRightString);
```

Compares a string (theLeftString) with a char pointer (theRightString). It returns a Boolean value of TRUE if the string is greater-than-or-equal to the char pointer.

Less-Than-Or-Equal Comparisons

The following three methods are less-than-or-equal operators. “Less-than” means that the first letter of one string is lower in the alphabet than the first letter of the other. *A* is lower in the alphabet than *B*.

```
friend BOOLEAN operator<=(const CString&
    theLeftString, const CString&
    theRightString);
```

Compares two strings, theLeftString and theRightString. This method returns a Boolean value of TRUE if theLeftString is less-than-or-equal to theRightString.

```
friend BOOLEAN operator<=(const char*
    theLeftString, const CString&
    theRightString);
```

Compares a char pointer (theLeftString) with a string (theRightString). It returns a Boolean value of TRUE if the char pointer is less-than-or-equal to the string.

```
friend BOOLEAN operator<=(const CString&
    theLeftString, const char* theRightString);
```

Compares a string (theLeftString) with a char pointer (theRightString). It returns a Boolean value of TRUE if the string is less-than-or-equal to the char pointer.

Indexing

```
char Get(int theIndex) const;
```

Given a position number (theIndex) in a string, returns the ASCII value of the character that occupies this position. If you pass in an index number beyond the scope of the string—for example, 100 for a string containing only 4 characters—the result is the ASCII value for a blank.

```
void Set(int theIndex, char theChar);
```

Sets the character at the given position in the string. theIndex is a position number, starting from zero (0). theChar is the ASCII value of the character. Suppose you want to set character number 100 in the string “hello” to a P. The result is a string that starts with *hello*, followed by 94 blanks that are followed by a P.

```
char& operator[](int theIndex);
```

Allows you to set and get positions. The string is always expanded when the index is out of range. The operator automatically expands the string, inserting blanks where there were no characters before. Do not store the address of the returned value.

Legal:	CString a;	Illegal:	CString a;
	a[100] = 'g';		a[3] = a[100];
	char c = a[3];		char &cr = a[5];
			char *cp = &(a[10]);

Input/Output Methods

```
friend ostream& operator<<(ostream& theOstream,
const CString& theString);
```

The output operator. Here is an example of usage:

```
CString S1="hello";
cout << S1;
```

```
friend istream& operator>>(istream& theIStream,
CString& theString);
```

The input operator. Here is an example of usage:

```
CString S1;
cin >> S1;
```

```
static void SetInputDelimiter(char
theDelimiter);
```

When characters are read from a stream into a string, there must be a delimiter that indicates where the reading is to stop. This delimiter is set to \n by default. This method allows you to set the delimiter for the input.

```
static char GetInputDelimiter(void);
```

Returns the delimiter that is used to indicate where reading is to stop when characters are read from a buffer into a stream.

```
static BOOLEAN ConsumesDelimiter(void);
```

Returns a Boolean value of TRUE if the delimiter symbol is included in the string and a value of FALSE if it is not.

```
static void ConsumeDelimiter(BOOLEAN doConsume);
```

Allows you to set whether the delimiter symbol is included in the string (TRUE) or not (FALSE). It is set to FALSE by default.

Other Non-operator Utilities

```
void Clear(void);
```

Clears the string.

```
int Length(void) const;
```

Returns the length of the string according to the actual number of characters it contains, regardless of the string's capacity.

Storage Growth Granularity

The following two methods pertain to the creation of new strings and to the “growth” of a string. “Growth” refers to the reallocation of memory to accommodate increases in a string’s size due to appends or concatenations. For example, if all strings had a default buffer size of 128 characters and you were to create the string “Hello”, a buffer size of 128 characters would be allocated for this five-character string. If you were to add 150 characters to this string, the buffer size would increase by another 128 characters for a storage capacity of 256 characters. That is, each time the limit is exceeded, the capacity for another 128 characters of memory is allocated. Perhaps you would prefer a much smaller or larger increment in buffer size. `SetBufferSize` allows you to specify the buffer size of a string in order to use memory efficiently.

```
static unsigned int GetBufferSize(void);
```

Returns an integer indicating the buffer allocation size in number of characters. If characters are added to the string so that it exceeds the storage capacity of the object, then the buffer increases by the integer amount returned.

```
static void SetBufferSize(unsigned int  
theNewBufferSize);
```

Allows the user to control the “growth” of a string by giving it a buffer allocation size.

Protected Methods

None.

Private Methods

```
void DestructiveHandle(void);
```

Takes care of cleanup.

```
void Grow(int theNewSize);
```

Takes care of sizing the internal structure of a string, according to theNewSize.

Summary: CString.h

```
#ifndef CString_H  
#define CString_H  
  
#include "PwrNames.h"  
#include "xvt.h"  
#include "CMem.h"
```

```

class ostream;
class istream;
class CStringBody;

class CString
{
public:
    // Class Utility:
    CString(void);
    CString(const char* theString);
    CString(char** theDisownedString);
    CString(const CString& theString);
    CString(int theResourceId, int theStringSize = 64);
    CString& operator=(const CString& theString);
    ~CString(void);

    // Conversion Operators: do not store, delete, or reallocate the
    // pointer returned. If you need a non-const char*, you may use
    // GetCharPtr(), but don't store or modify its return value.

    operator const char*(void) const;
    char* GetCharPtr(void) const;

    // For safe (but more expensive) conversion into a C string use
    char* GetString(void) const;

    // Catenating (multiply overloaded to avoid ambiguity):
    CString& operator+=(const CString& theStringToCatenate);
    CString& operator+=(const char* theStringToCatenate);

    friend CString operator+(const CString& theLeftString,
                             const char* theRightString);
    friend CString operator+(const char* theLeftString,
                             const CString& theRightString);
    friend CString operator+(const CString& theLeftString,
                             const CString& theRightString);

    void Append(const CString& theString, int theNumberOfCharacters);
    void Append(const char* theString, int theNumberOfCharacters);

    // Comparison (multiply overloaded to avoid ambiguity):
    friend BOOLEAN operator==(const CString& theLeftString,
                              const CString& theRightString);
    friend BOOLEAN operator==(const CString& theLeftString,
                              const char* theRightString);
    friend BOOLEAN operator==(const char* theLeftString,
                              const CString& theRightString);

    friend BOOLEAN operator!=(const CString& theLeftString,
                              const CString& theRightString);
    friend BOOLEAN operator!=(const CString& theLeftString,
                              const char* theRightString);
    friend BOOLEAN operator!=(const char* theLeftString,
                              const CString& theRightString);

    friend BOOLEAN operator>(const CString& theLeftString,
                             const CString& theRightString);
    friend BOOLEAN operator>(const char* theLeftString,
                             const CString& theRightString);
    friend BOOLEAN operator>(const CString& theLeftString,
                             char* theRightString);

```

```

friend BOOLEAN operator<(const CString& theLeftString,
    const CString& theRightString);
friend BOOLEAN operator<(const char* theLeftString,
    const CString& theRightString);
friend BOOLEAN operator<(const CString& theLeftString,
    const char* theRightString);

friend BOOLEAN operator>=(const CString& theLeftString,
    const CString& theRightString);
friend BOOLEAN operator>=(const char* theLeftString,
    const CString& theRightString);
friend BOOLEAN operator>=(const CString& theLeftString,
    const char* theRightString);

friend BOOLEAN operator<=(const CString& theLeftString,
    const CString& theRightString);
friend BOOLEAN operator<=(const char* theLeftString,
    const CString& theRightString);
friend BOOLEAN operator<=(const CString& theLeftString,
    const char* theRightString);

// Indexing - the safe way:
char Get(unsigned int theIndex) const;
void Set(unsigned int theIndex, char theChar);

// ... and the mostly safe way (don't store the address returned):
char& operator[](unsigned int theIndex);

// Input, Output:
friend ostream& operator<<(ostream& theOstream,
    const CString& theString);
friend istream& operator>>(istream& theIStream,
    CString& theString);

static void SetInputDelimiter(char theDelimiter);
static char GetInputDelimiter(void);

static BOOLEAN ConsumesDelimiter(void);
static void ConsumeDelimiter(BOOLEAN doConsume);

// Other non-operator utilities:
void Clear(void);
int Length(void) const;

// Storage Growth granularity
static unsigned int GetBufferSize(void);
static void SetBufferSize(unsigned int theNewBufferSize);

private:

    CStringBody *itsBody;

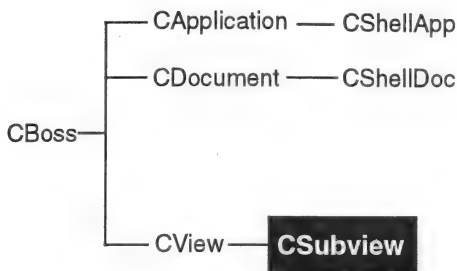
    static unsigned int BUFSIZE;
    static char itsDelimiter;
    static BOOLEAN itConsumesDelimiter;

    void DestructiveHandle(void);
    void Grow(int theNewSize);
};

#endif CString_H

```

CSubview



Description

CSubview objects are special types of CView objects that have the added ability to contain other nested CSubview objects.

A CSubview object can contain other views within it. Any actions, events, and changes it receives are recursively filtered down to all of the subview's child views. If a subview cannot handle an event that has been passed to it, the subview passes the event back up to its enclosing view.

The concept of nested views and enclosures is crucial in XVT-Power++. All CSubview objects except for windows have an object that acts as the enclosure. CSubview objects can nest inside windows—which are themselves a type of subview—as well as inside other views. Each enclosing view establishes its own local coordinate system and clipping properties. Nesting views inside a subview has a stacking effect. When subviews are added to a view, the last one added is the topmost view on the stack.

A common error made by novices to XVT-Power++ is to distinguish between CSubview and its parent class CView. The only difference between the classes is that subviews can contain other nested views. This means that any CView object can itself be nested inside of a CSubview object. In the class hierarchy, the CSubview class is the *child* of the CView class, whereas in the object hierarchy a CSubview object is the *enclosure* of a CView object.

Heritage

Superclass: CView

Subclasses: CGrid, CIcon, CShape, CSketchPad, CVirtualFrame, CWindow

Usage

The most important functionality implemented at this level of the XVT-Power++ class hierarchy is that of event filtering. CSubview contains two different types of event handling methods: the Event and DoEvent methods. To understand the difference, consider the following two event methods:

- Draw — draw this CSubview object.
- DoDraw — draw this CSubview object *and* inform all of its subviews to draw.

Event methods are handled by the CSubview object itself, while DoEvent methods are both handled and passed down to the rest of the subviews, which, in turn, pass them on down to any subviews they may contain.

Public Data Members

None.

Protected Data Members

CList*	itsSubviews;	list of the views it contains
--------	--------------	-------------------------------

Private Data Members

CView*	itsSelectedView;	the view that should receive events first
CView*	itsKeyFocus;	the view that receives the keyboard event
CList	itsSelectedViews;	list of selected views

Public Methods

Constructor and Destructor Methods

```
CSubview(CSubview* theEnclosure, const CRect& theRegion);
```

A constructor. theEnclosure is a pointer to the subview that will contain the subview. theRegion is a coordinate location, local to the enclosure, that is used to place the subview.

```
CSubview(const CSubview& theSubview);
```

A copy constructor that creates a new subview with the same enclosure, visibility attributes, enabled/disabled attributes, environment, and so on as the original subview. However, the views nested within the subview are not copied.


```
CSubview& operator=(const CSubview&  
theSubview);
```

An assignment operator. It copies the attributes of the original subview, creating a new view that has the same color, glue, environment, visibility state, and so on. However, the views nested within the subview are not copied.

```
virtual ~CSubview(void);
```

The destructor. When a subview is deleted, the destructor also deletes all of the views that are still nested inside of it. Because all nested views are deleted, you do not have to delete them yourself anywhere else. In fact, if you write a CSubview-derived class, be sure not to delete any views nested inside the new class's destructor. Allow these views to be deleted by the inherited CSubview destructor.

Visibility State Methods

The following two methods are present at the CView level, in addition to Hide and Show, which actually take care of hiding and showing the view. For CView, DoShow and DoHide are present so that XVT-Power++ can achieve a "wide interface" that treats CView and CSubview objects in the same way. At the CSubview level, DoShow and DoHide ensure that the Hide and Show operations are passed on to all of the recursively nested subviews. You must still generate a DoDraw message to see the results.

```
virtual void DoShow(void);
```

Sends the subview a message to become visible; the subview reveals itself and in turn sends a message to its subviews to reveal themselves. This message is passed recursively until it reaches the deepest subview. This method does not update the window; thus, you may need to follow it by a call to DoDraw.

```
virtual void DoHide(void);
```

Sends the subview a message to become invisible; the view hides itself and sends a message to its subviews to hide themselves. This message is passed recursively until it reaches the deepest subview. This method does not update the window; thus, you may need to follow it by a call to DoDraw.

Activate Events

The following two methods are present at the CView level in addition to Activate and Deactivate, which actually take care of activating and deactivating the view. For CView, DoActivate and DoDeactivate are present so that XVT-Power++ can achieve a

“wide interface” that treats CView and CSubview objects in the same way. At the CSubview level, DoActivate and DoDeactivate ensure that the Activate and Deactivate operations are passed on to all of the recursively nested subviews. The events passed are downward-chaining.

virtual void DoActivate(void);

Changes a subview’s active state so that it is activated. The subview in turn notifies its subviews to become active. This message may be passed down until it reaches the deepest subview.

virtual void DoDeactivate(void);

Changes a subview’s state so that it is deactivated. The subview in turn notifies its subviews to become inactive. This message may be passed down until it reaches the deepest subview.

Enable/Disable Events

virtual void DoEnable(void);

Enables a subview so that it can receive events. The subview in turn notifies its subviews to become enabled. This message may be passed down until it reaches the deepest subview.

virtual void DoDisable(void);

Disables a subview so that it cannot receive events. The subview in turn notifies its subviews to become disabled. This message may be passed down until it reaches the deepest subview.

Mouse Events

Mouse events are passed directly to the view at the location where the event occurs. Each of the following methods has a `theButton` argument that specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right). By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button.

The base mouse methods, which reside at the CView level, have an argument called `theLocation`, which marks the location of the cursor when the mouse button was clicked. This point, `theLocation`, is in *local* coordinates, that is, coordinates *relative* to the view’s top-left corner. Like the base mouse methods, the “Do” mouse methods take a parameter named `theLocation`. However, for the “Do” methods `theLocation` is in *global*, window-relative coordinates. The “Do” methods take the global coordinate, translate it into a local coordinate, and call the corresponding base mouse

method. However, at the CSubview level, more computation is required to verify whether the global coordinate is contained in one of the views.

```
virtual void DoMouseDouble(CPoint theLocation,
                           short theButton    = 0,
                           BOOLEAN isShiftKey  = FALSE,
                           BOOLEAN isControlKey= FALSE);
```

When the user double clicks the mouse at global coordinate theLocation, the subview translates this coordinate into a local coordinate and passes the MouseDouble event to its selected view. If no view is selected, it searches for its deepest view at theLocation, which handles the event.

```
virtual void DoMouseDown(CPoint theLocation,
                          short theButton    = 0,
                          BOOLEAN isShiftKey  = FALSE,
                          BOOLEAN isControlKey= FALSE);
```

When the user presses down a mouse button at global coordinate theLocation, the subview translates this coordinate into a local coordinate and passes the MouseDown event to its selected view. If no view is selected, it searches for its deepest view at theLocation, which handles the event.

```
virtual void DoMouseMove(CPoint theLocation,
                          short theButton    = 0,
                          BOOLEAN isShiftKey  = FALSE,
                          BOOLEAN isControlKey= FALSE);
```

When the user moves the mouse at global coordinate theLocation, the subview translates this coordinate into a local coordinate and passes the MouseMove event to its selected view. If no view is selected, it searches for its deepest view at theLocation, which handles the event.

```
virtual void DoMouseUp(CPoint theLocation,
                       short theButton    = 0,
                       BOOLEAN isShiftKey  = FALSE,
                       BOOLEAN isControlKey= FALSE);
```

When the user releases a mouse button at global coordinate theLocation, the subview translates this coordinate into a local coordinate and passes the MouseUp event to its selected view. If no view is selected, it searches for its deepest view at theLocation, which handles the event.

Draw Methods

```
virtual void DoDraw(void);
```

The view draws itself and notifies each of its subviews to do the same. By default the clipping region is set to the entire region

that the view occupies. If you do not want to calculate a clipping region and just want to draw the entire view, then you can call this method without passing anything.

```
virtual void DoDraw(const CRect&  
theClippingRegion);
```

Takes care of any drawing the subview must do and draws all of the its subviews, ensuring that all of the subviews are clipped to the enclosing subview. `theClippingRegion` is the part of the subview that needs to be refreshed, and it is in global, window-relative coordinates. This method is automatically called after an `E_UPDATE` event.

Printing Methods

```
virtual void DoPrintDraw(const CRect&  
theClippingRegion);
```

A method that is equivalent to `DoDraw`. It goes through the view's list of subviews notifying them to print. In response, do any drawing that should be sent to the printer and notify their subviews (if any) to `DoPrintDraw`. This drawing is just like the drawing that is done on the screen using the XVT Portability Toolkit `win_draw` functions. If you were to write your own printer `DoDraw` method, it would look just as if you were printing from the screen. You would still print using the view's XVT Portability Toolkit window, which can be obtained through the `CWindow GetXVTWindow` method. `GetXVTWindow` can return either a regular screen window or a printed window, depending on whether printing is being done. `DoPrintDraw` just calls the regular `DoDraw` method, and in most cases this is adequate. For some views, however, you may want to modify the way that drawing is done for the printer and not for the screen. In this case, you can override `DoPrintDraw`, putting in your own drawing specifications.

Coordinates and Locations

```
virtual void DoSetOrigin(const CPoint&  
theDeltaPoint);
```

Takes the current origin of the view and adds `theDeltaPoint` to it. The view in turn sends a message to each of its subviews to add `theDeltaPoint` to its current origin. `theDeltaPoint` is not an absolute origin but rather a difference that is added to the origins of all of the nested subviews—a delta. For example, if you add 5 pixels to the origin of the subview, then you must add 5 pixels to the origins of all the subviews recursively nested within it. This is a convenient and safe way to change the

origins of nested subviews without altering their relationships with one another.

Environment Methods

```
virtual void SetEnvironment(const CEnvironment&  
theNewEnvironment, BOOLEAN isUpdate =  
FALSE);
```

Sets the environment for a subview, using theNewEnvironment that is passed to it. By default, subviews share their enclosure's environment. However, as soon as you use SetEnvironment to give a subview an environment of its own, the subview uses that environment instead of the shared environment.

When the global environment is changed, all views sharing that environment get a SetEnvironment message with the isUpdate parameter set to TRUE. The isUpdate parameter indicates whether this SetEnvironment message is simply an update message or whether it is a "real" SetEnvironment message meaning that this particular subview should set its own environment to the new environment.

```
virtual void DoSetEnvironment(const  
CEnvironment& theNewEnvironment, BOOLEAN  
isUpdate = FALSE);
```

Sets the environment for the CSubview object, which, in turn, notifies each of its subviews of the environment parameters of theNewEnvironment. Bewary about calling DoSetEnvironment because most subviews share an environment with their enclosure. If they are sharing, you can set the enclosure's environment as often as you want, which automatically sets the environment of its subviews. However, DoSetEnvironment not only sets the environment of views sharing the environment but also the environment of all other nested views.

When a shared environment is changed, all views sharing that environment get a DoSetEnvironment message with the isUpdate parameter set to TRUE. The isUpdate parameter indicates whether this DoSetEnvironment message is simply an update message or whether it is a "real" DoSetEnvironment message meaning that this particular view should set its own environment to the new environment and pass this change to all of its child views who are sharing its environment.

```
virtual void SetFont(const FONT& theNewFont,  
BOOLEAN isUpdate = FALSE);
```

Takes a font (theFont) and sets the subview's environment to have that font. The isUpdate parameter indicates whether this

SetFont message is simply an update message or whether it is a “real” SetFont message meaning that this particular view should set its own font to the new font.

```
virtual void DoSetFont(const FONT& theNewFont,  
    BOOLEAN isUpdate = FALSE);
```

Takes a font and sets the subview’s environment to have that font. It recursively passes theNewFont to all of its subviews.

When the font of a shared environment is changed, all views sharing that environment get a DoSetFont message with the isUpdate parameter set to TRUE. The isUpdate parameter indicates whether this DoSetFont message is simply an update message or whether it is a “real” DoSetFont message meaning that this particular view should set its own font to the new font and pass this change to all of its child views who are sharing its environment.

Glue Methods

```
virtual void DoSetGlue(GLUETYPE theGlue);
```

Sets the glue type of the subview and then recursively sets the glue of the its subviews to that type.

Sizing and Dragging Methods

```
virtual void DoSetDragging(BOOLEAN  
    isDraggable);
```

Takes a Boolean value that sets the dragging of a view to TRUE so that the subview can be dragged with the mouse or to FALSE so that the subview cannot be dragged. It then recursively sets the dragging for the subview’s child views.

```
virtual void DoSetSizing(BOOLEAN isSizable);
```

Takes a Boolean value that sets the sizing of a subview to TRUE so that the subview can be sized or to FALSE so that the subview cannot be sized. It then recursively sets the sizing for the subview’s child views.

Selection Methods

```
virtual void SetSelectedView(CView*  
    theSelectedView);
```

Sets the selected view, which is the view that automatically receives all events. If multiple views are selected, theSelectedView is the last view that was selected.

```
virtual CView* GetSelectedView(void) const;
```

When a view is selected, the window containing this view returns a pointer to it. The view can thus be moved and sized by the user. If there is no currently selected subview, this method returns NULL. If multiple views are selected, GetSelectedView returns the last view that was selected.

```
virtual void AddSelectedView(CView* theView);
```

A method that is called automatically when a view nested inside the subview becomes selected. It adds theView to the subview's list of selected views.

```
virtual void RemoveSelectedView(CView* theView);
```

A method that is called automatically when a view nested inside the subview becomes deselected. It removes theView from the subview's list of selected views.

```
virtual const CList* GetSelectedViews(void);
```

Returns a list of the subview's selected views.

Utility Methods

```
virtual void PlaceTopSubview(const CView* theView);
```

In the case of overlapping views, places theView at the top of the stack.

```
virtual void PlaceBottomSubview(const CView* theView);
```

In the case of overlapping views, places theView at the bottom of the stack.

```
virtual CView* FindSubview(const CPoint& theLocation) const;
```

Returns a pointer to the top-level subview that contains the global, window-relative coordinate theLocation.

```
virtual CView* FindSubview(int theViewId) const;
```

Given the ID number of a subview, theViewId, searches a list of subviews and returns a pointer to the subview whose ID matches theViewId.

```
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
```

In a nesting of views, finds the *deepest* subview that contains the global, window-relative coordinate theLocation.

```
virtual void FindSubviews(const CPoint&
    theLocation, CList* theList) const;
```

Returns a list of every subview, nested or overlapping, that contains the global, window-relative coordinate theLocation.

```
virtual const CList* GetSubviews(void) const;
```

Returns a pointer to the subview's list of subviews.

```
int GetNumViews(void) const;
```

Returns the number of child views on the subview's list of subviews.

```
virtual const CList* GetSubObjects(void) const;
```

A pure virtual method that must be defined by any class derived from CBoss, specifically, CApplication, CDocument, CView, and CSubview. It returns a list of any subobjects associated with a given object. For example, the application would return a list of all its documents, a document would return a list of all its windows, a window would return a list of all its enclosed views, a subview would return a list of all its enclosed views, and so on.

Command and Key Methods

```
virtual void SetKeyFocus(CView
    *theFocusedView);
```

Sets the view that is to receive the keyboard event (theFocusedView). This is necessary because, unlike mouse input which uses the mouse cursor to point to a specific screen coordinate, keyboard input does not clearly point to the view to which it is directed.

```
CView* GetKeyFocus(void) const;
```

Returns the view to which the keyboard input is currently directed. If there is no such view, this method returns NULL.

Methods for Adding and Removing Subviews

The following methods are called internally by XVT-Power++ classes.

```
virtual void AddSubview(const CView*
    theSubview);
```

Adds a pointer to a view, theSubview, to the view's list of views.


```
virtual void RemoveSubview(const CView*
thesubview);
```

Removes a view, thesubview, from the subview's list of views.

Other Methods

```
virtual void DoKey(int theKey, BOOLEAN
isShiftKey, BOOLEAN isControlKey);
```

A method that is called when there is a keyboard event. When the user presses a key on the keyboard, the XVT Portability Toolkit sends an E_CHAR event to the switchboard, which passes this event to the selected window. The window checks to see if one of its subviews is selected, and, if so, passes the event to the appropriate subview. If no subview is selected, the keyboard event propagates upwards from the window to the document and finally to the application. A keyboard event goes directly to the application if no windows are open.

theKey is the ASCII number of the character key that was pressed; isShiftKey and isControlKey indicate whether the SHIFT key or CONTROL key was pressed in conjunction with the character key. You can override DoKey to do something specific to a keyboard action within the application. The default mechanism does nothing.

```
virtual void DoSize(const CRect& theNewSize);
```

Sizes the subview and recursively updates the origin of all of its subviews in case the sizing of the view involved some moving. It also calls upon each of the subviews' glue objects, informing them of the new size so that they can, according to their type, resize their owners by stretching them, moving them, shrinking them, or whatever the glue has been set to do.

```
virtual CPoint AutoScroll(UNITS
theHorizontalChange, UNITS
theVerticalChange);
```

Automatically scrolls the subview if it is inside a scroller or other view that can be autoscrolled. theHorizontalChange takes a number of pixels. If the number is zero (0), no autoscrolling occurs; if the number is positive, the subview scrolls to the right; if it is negative, the view scrolls to the left. theVerticalChange also takes a number of pixels. If the number is zero (0), no autoscrolling occurs; if the number is positive, the view scrolls downward; if it is negative, the view scrolls upward.

```
CView* FindEventTarget(const CPoint&
theLocation) const;
```

A method that is called to find the target of a mouse event within a subview. This virtual method is defined to return the deepest subview at the global coordinates of theLocation. This method can be overridden by a subview in order to trap all events regardless of the subviews it contains or to change the event targeting algorithm. For example, a list box contains many subviews (text items). When you click on a text item inside a list box, the event does not go to the CText view. Instead, the list box overrides FindEventTarget and traps the event by returning this (itself) as the target for the mouse events received. Then, it treats the mouse events as necessary, without sending them on to its subviews.

Protected Methods

```
BOOLEAN ISubview(BOOLEAN isVisible, GLUETYPE
theGlue);
```

The initializer, which takes a visibility state and a glue type.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
```

```

virtual void Enable(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);

```

```

virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

Summary: CSubview.h

```

#ifndef CSubview_H
#define CSubview_H

#include "xvt.h"
#include "CView.h"

class CSubview : public CView
{
    friend class CView;
    friend CNativeView;

public:
    // Construct
    CSubview(CSubview* theEnclosure, const CRect& theRegion);
    CSubview(const CSubview& theSubview);
    CSubview& operator=(const CSubview& theSubview);
    virtual ~CSubview(void);

    // visibility state:
    virtual void DoShow(void);
    virtual void DoHide(void);

    // activate events
    virtual void DoActivate(void);
    virtual void DoDeactivate(void);

    // enable/disable events
    virtual void DoEnable(void);
    virtual void DoDisable(void);

```

```

// Mouse Events:
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);

// Drawing:
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoDraw(void);

// Other Events:
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoSize(const CRect& theNewSize);

// Coordinates and locations:
virtual void DoSetOrigin(const CPoint& theDeltaPoint);

// Environment:
virtual void SetEnvironment(const CEnvironment& theNewEnvironment,
    BOOLEAN ISuPDATE = false);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment,
    BOOLEAN isUpdate = FALSE);

virtual void SetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont,
    BOOLEAN isUpdate = FALSE); // Glue:

//Glue:
virtual void DoSetGlue(GLUETYPE theGlue);

// Sizing and dragging:
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetSizing(BOOLEAN isSizable);

virtual void AddSelectedView(CView* theView);
virtual void RemoveSelectedView(CView* theView);
virtual const CList* GetSelectedViews(void);

// Utility:
virtual void PlaceTopSubview(const CView* theView);
virtual void PlaceBottomSubview(const CView* theView);
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation,CList* theList)
    const;

virtual const CList* GetSubviews(void) const;

virtual void SetSelectedView(CView* theSelectedView);
virtual CView* GetSelectedView(void) const;

int GetNumViews(void) const;

// Command and key events
virtual void SetKeyFocus(CView *theFocusedView);
CView* GetKeyFocus(void) const;

```

```
// Subview addition and removal:
virtual void AddSubview(const CView* theSubview);
virtual void RemoveSubview(const CView* theSubview);

// other events:
virtual CPoint AutoScroll(UNITS theHorizontalChange,
                          UNITS theVerticalChange);

CView* FindEventTarget(const CPoint& theLocation) const;

protected:
    CList* itsSubviews; // list of the views it contains
    BOOLEAN ISubview(BOOLEAN isVisible, GLUETYPE theGlue);

private:
    CView* itsSelectedView; // the view which should receive events first
    CView* itsKeyFocus;     // the view that receives the key event.
};

#endif CSubview_H
```

CSwitch

CMem	CGlue	CSwitchBoard
CError	CDesktop	CStartup
	CEnvironment	CGlobalClassLib
	CGlobal	CGlobalUser
		CResourceMgr
		CPrintMgr

Description

CSwitchBoard is a utility class that serves as a liason between XVT Portability Toolkit and XVT-Power++, providing an interface between all the objects and all of the events in the system.

The switchboard channels all events in the system to the appropriate object. For example:

- a termination or start-up event goes to the application object.
- a resize event goes to the appropriate window object.
- a key event is channeled to the appropriate view.

The switchboard *automatically* maps the XVT Portability Toolkit events to their corresponding XVT-Power++ methods, so a user does not need to deal with these events. However, if you are interested in how this mapping is done, the following discussion describes it. The purpose here is not to define the XVT Portability Toolkit events but rather to describe them just enough that the switchboard mappings make sense. For further details on XVT Portability Toolkit events, consult the *XVT Programmer's Guide*.

Usage

The switchboard is created automatically, and its use is hidden from the programmer.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

<code>static CApplication*</code>	<code>itsApplication;</code>	pointer to the application object
<code>static BOOLEAN*</code>	<code>itIsTextEvent;</code>	interface to text-event system
<code>static BOOLEAN</code>	<code>itIsInstantiated;</code>	assures only one instance
<code>static int</code>	<code>itIsUpdateEvent;</code>	whether the handler is managing an update

Public Methods

```
static long EventHandler(WINDOW win, EVENT *ep);
```

Acts as a global event handler for all window objects and dispatches each event to the appropriate window, if necessary.

```
static long DialogHandler(WINDOW win, EVENT *ep);
```

Acts as a global event handler for all dialog window objects and dispatches each event to the appropriate dialog window, if necessary.

Protected Methods

None.

Private Methods

Constructor and Destructor Methods

```
CSwitchBoard(CApplication* theApplication);
```

A constructor.

```
CSwitchBoard(const CSwitchBoard& theSwitchBoard);
```

A copy constructor.

```
~CSwitchBoard(void);
```

The destructor.

Summary:CSwitchBoard.h

```
#include "xvt.h"

class CDesktop;
class CApplication;

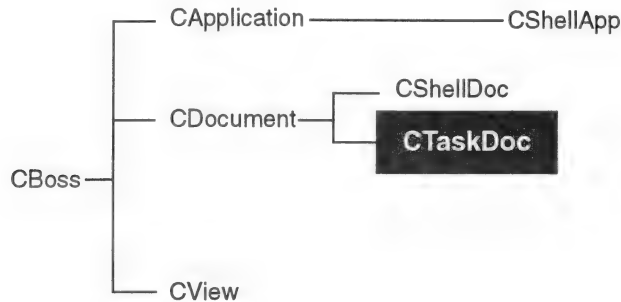
class CSwitchBoard
{
    public:
        // Window event handlers
        static long EventHandler(WINDOW win, EVENT *ep);
        static long DialogHandler(WINDOW win, EVENT *ep);

    private:
        friend class CApplication;

        // constructor and destructor
        CSwitchBoard(CApplication* theApplication);
        CSwitchBoard(const CSwitchBoard& theSwitchBoard);
        ~CSwitchBoard(void);

        // the switchboard data
        static CApplication* itsApplication; // pointer to the application
                                                // object
        static BOOLEAN* itIsTextEvent;       // interface to text-event system
        static BOOLEAN itIsInstantiated;     // assure only one instance
        static int itIsUpdateEvent;          // is the handler managing an
                                                // update
};
```

CTaskDoc



Description

CTaskDoc is a class that interacts with the task window, which is a window that encloses the entire application on some platforms. On most platforms, the task window is an abstract window; that is, there is no visible window on the screen. The appearance of the task window depends on the platform. A CTaskDoc object is a document that instantiates the task window and functions as the task window's document.

Heritage

Superclass: CDocument

Usage

This class is instantiated only by CApplication.

Friends

```
friend class CApplication;
```

Public Methods

Inherited Methods

```
virtual void BuildWindow(void);
```

An overridden method that specifies how to create the task window.

Protected Methods

None.

Private Methods

Constructor

```
CTaskDoc(CApplication *theApplication,int
theId);
```

A constructor. It is a private method because no one can create the document for the task window. The CApplication object actually creates the document. theApplication is a pointer to the application object that creates the document, and theId is the ID number of the document.

Methods Inherited From CDocument But Not Overridden

Methods are listed alphabetically by name.

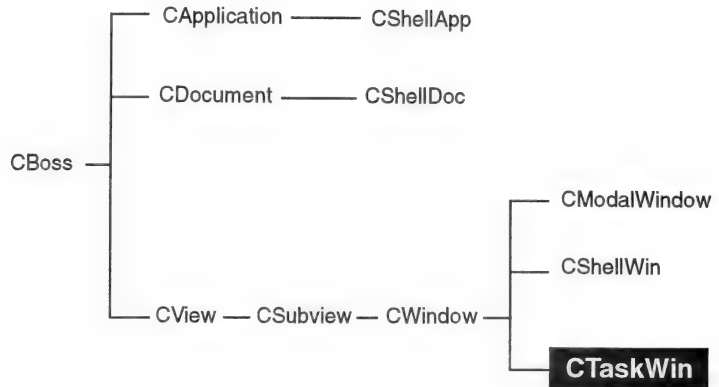
```
virtual void AddWindow(CWindow * theWindow);
virtual void ChangeFont(FONT theFont,FONT_PART thePart);
virtual BOOLEAN DoClose(void);
virtual void CloseAll(void);
virtual void DoKey(int theKey, BOOLEAN theShiftKey,BOOLEAN theControlKey);
virtual void DoMenuCommand(MENU_TAG theMenuItem,BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual BOOLEAN DoPageSetUp(void);
virtual BOOLEAN DoPrint(void);
virtual BOOLEAN DoSave(void);
virtual BOOLEAN DoSaveAs(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CWindow* FindWindow(int theId) const;
int GetId(void) const;
int GetNumWindows(void);
virtual const CList* GetSubObjects(void) const = NULL;
virtual void SetId(int theId);
BOOLEAN GetSave(void) const;
virtual void RemoveWindow(CWindow *theWindow);
virtual void SetEnvironment(const CEnvironment& theEnvironment);
void SetSave(BOOLEAN isSaved);
virtual void UpdateMenus(void);
```

Summary: CTaskDoc.h

```
#ifndef CTaskDoc_HXX
#define CTaskDoc_HXX
```

```
#include "PwrDef.h"
#include CDocument_i
class CTaskDoc : public CDocument
{
    public:
        virtual void BuildWindow(void);
    private:
        friend class CApplication;
        CTaskDoc(CApplication *theApplication,int theId);
};
#endif CTaskDoc_HXX
```

CTaskWin



Description

CTaskWin is a class that interacts with the task window, which is a window that encloses the entire application on some platforms. On most platforms, this is an abstract window; that is, there is no visible window on the screen. The appearance of the task window depends on the platform. For example, on Motif when no other windows are open, the task window looks like a small floating menubar. On the Macintosh, the task window appears as a menubar across the top of the screen. On Windows and Presentation Manager, the task window is an actual window that contains all of the other windows in the application. CTaskWin is one of two XVT-Power++ classes that interact with the task window. The other is CTaskDoc.

Heritage

Superclass: CWindow

Usage

This class is instantiated only by CTaskDoc.

Environment

Friends

friend class CTaskDoc;

Public Methods

None.

Protected Methods

None.

Private Methods

Constructor

```
CTaskWin(CDocument *theDocument, WINDOW
theXVTWindow);
```

A constructor. It is a private method because the task window can be created only by the CTaskDoc object. `theDocument` is a pointer to the document object that creates the window, and `theXVTWindow` is a handle to the XVT Portability Toolkit's task window.

Inherited Methods

```
virtual void SizeWindow(int theNewWidth, int
theNewHeight);
```

A method that is called internally when the window is sized by the user. The window can be sized only on certain platforms.

```
virtual void ChangeFont(FONT theNewFont,
FONT_PART thePart);
```

This method changes the font of the task window only on Windows and Presentation Manager platforms. On these two platforms, `ChangeFont` works as it does for any other window, as explained in the following paragraphs. On other platforms, `ChangeFont` is disabled for the task window and is propagated upwards to the application.

When the user selects a font from the menubar, `ChangeFont` sends a font object, `theFont`, that indicates the current state of the Font/Style menu. The specific font property that was changed is indicated in `thePart`, which is of type `FONT_PART`, an XVT Portability Toolkit variable that is defined as follows:

```
typedef enum{          /* symbol for part of font desc.*/
F_STYLE;              /* style (bold, italic, ect.)*/
F_FAMILY;             /* FAMILY (Times, etc.) */
F_SIZE;               /* point size part*/
} FONT_PART;
```

The window verifies whether it has a selected view, and if it does, it sends the event to the selected view. If not, it sends the event to itself—only if this window actually has an environment

of its own; if the window is sharing an environment with its document, `ChangeFont` propagates the event up to the document level. This event goes directly to the application if no windows are up.

virtual BOOLEAN Close(void);

Closes the window, which shuts down the application. This method is called when XVT-Power++ closes the task window. Only XVT-Power++ can close the task window.

virtual void Enable(void);

Enables the window so that it can receive events. The task window can be enabled only on Windows platforms.

virtual void Disable(void);

Disables the window so that it cannot receive events. The task window can be disabled only on Windows platforms.

virtual void Show(void);

Makes the window visible. This method works only on Windows platforms.

virtual void Hide(void);

Makes the window invisible. This method works only on Windows platforms.

virtual void UpdateUnits(CUnits* theUnits);

Updates the `CUnits` object indicated by `theUnits`, which is the `CUnits` object owned by the `CTaskWin` object. See `CUnits`.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

`virtual CUnits* GetUnits(void) const;`
`virtual void SetUnits (CUnits* theCoordinateUnits);`

From CView

`virtual void Activate(void);`
`virtual void Deactivate(void);`
`virtual BOOLEAN DoPrint(const CRect& theRegion) const;`
`virtual void DoTimer(long theTimerId);`
`virtual void DoUser(long theUserId, void* theData);`
`virtual void Draw(void);`

```

virtual CView* FindHitView(const CPoint& theLocation) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);

```



```

virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;

```

```

virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjeects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);

```

From CWindow

```

virtual void DoActivateWindow(void);
virtual void DoDeactivateWindow(void);
void DoHScroll(SCROLL_CONTROL theEvent, short thePos);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
void DoVScroll(SCROLL_CONTROL theEvent, short thePos);
virtual void Draw(const CRect& theClippingRegion);
virtual CRect GetClippedFrame(void) const;
virtual CDocument* GetDocument(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual WINDOW GetXVTWindow(void) const;
virtual BOOLEAN IsBackgroundDrawn(void) const;
virtual BOOLEAN IsClosable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void SetBackgroundDrawing(BOOLEAN isBackgroundDrawn);
virtual void SetEnclosure(CSubview* theEclosure);
virtual void SetSelectedView(CView *theSubview);
virtual void SetTitle(const CString& theNewTitle);
virtual void Size(const CRect& theNewSize);
virtual void UpdateMenus(void);

```

Summary: CTaskWin.h

```

#ifndef CTaskWin_HXX
#define CTaskWin_HXX

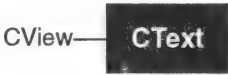
```

```
#include "CWindow.h"

class CTaskWin : public CWindow
{
    private:
        friend class CTaskDoc;
        CTaskWin(CDocument *theDocument, WINDOW theXVTWindow);
        // Inherited methods:
        virtual void SizeWindow(int theNewWidth, int theNewHeight);
        virtual void ChangeFont(FONT theNewFont, FONT_PART thePart);
        virtual BOOLEAN Close(void);
        virtual void Enable(void);
        virtual void Disable(void);
        virtual void Show(void);
        virtual void Hide(void);
        virtual void UpdateUnits(CUnits* theUnits);
};

#endif CTaskWin_HXX
```

CText



Description

CText is a static text drawing class. Also, it is the only text object in XVT-Power++ that can be set in color instead of black and white. Use it instead of the more complete NTextEdit or NLineText when you simply need to display a static, read-only line of text in a window. Examples of use include titles, one line instructions, and button names. If you need to display multiple lines or paragraphs, use the NTextEdit class (which can be specified to be read only).

Heritage

Superclass: CView

Usage

Create a CText object, initialize it, and register it.

Environment

Text is drawn in the foreground color, which you can set. The background for text is either transparent (whatever was already there shows through) or opaque. If the text is drawn opaquely, the spaces between and around the letters are drawn in the background color, which you can set. Also, CText has a method for setting its font.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

BOOLEAN	itIsOpaque;	whether the background is transparent or opaque
BOOLEAN	itIsSelected;	whether the text is selected
int	itsAscent;	drawing distance from top

Public Methods

Constructor, Destructor, and Initializer Methods

```
CText(CSubview* theEnclosure, const CPoint&
      theTopLeft,
      const CString theText = NULLString);
```

The constructor. It takes an enclosure, which is the view that will contain the line of text. theTopLeft specifies the coordinate at which the line of text starts. Finally, theText is the text string to be inserted into the CText object.

```
CText(const CText& theText);
```

A copy constructor that duplicates the attributes of a CText object, as well as the text string being displayed.

```
CText& operator=(const CText& theText);
```

An assignment operator that duplicates the attributes of a CText object, as well as the text string being displayed.

```
~CText(void);
```

The destructor, which cleans up the text edit system.

```
BOOLEAN IText(const CString& theText, BOOLEAN
               isOpaque= FALSE, GLUETYPE theGlue =
               NULLSTICKY);
```

The initializer. It takes a new text string, a visibility state, and a glue type.

Setting and Getting Text

```
virtual void SetText(const CString& theText);
```

Sets the text inside a text object to the string that is passed in.

```
virtual void SetTitle(const CString& theText);
```

Sets the text inside a text object to the string that is passed in.

```
virtual const CString GetText(void) const;
```

Returns the string of text that is being stored inside a text object.

```
virtual BOOLEAN IsOpaque(void) const;
```

Returns a Boolean value of TRUE if the background for the text is opaque and FALSE if it is transparent (whatever was already there shows through). If the background is opaque, then it is drawn in the color specified by the text object's environment. The text itself is drawn in the foreground color that is set by the environment.

```
virtual void SetOpaque(BOOLEAN isOpaque);
```

Sets whether the text object's background is opaque. A value of TRUE means that the background is opaque and is therefore drawn in the color specified by the text object's environment.

Selecting Text

```
virtual void Select(void);
```

Selects the text inside a text object, causing it to become inverted.

```
virtual void Deselect(void);
```

Deselects the text inside a text object so that it is no longer inverted.

```
virtual BOOLEAN IsSelected(void) const;
```

Returns a Boolean value of TRUE if the text inside a text object is selected and a value of FALSE if it is not.

Sizing Utilities

```
virtual UNITS GetHeight(void) const;
```

Returns the height, in pixels, of the text object, which is determined by its font.

```
virtual UNITS GetWidth(void) const;
```

Returns the width, in pixels, of the text object.

Inherited Utilities

```
virtual void Draw(const CRect&  
theClippingRegion);
```

Takes care of any drawing the text object must do. theClippingRegion, which is the portion of the text object to be drawn, is represented in global, window-relative coordinates. If theClippingRegion is set to NULL, the entire text object is drawn.

```
virtual void SetFont(const FONT& theFont,  
BOOLEAN isUpdate = FALSE);
```

Takes a font (theFont) and sets the text box's environment to have that font. The isUpdate parameter indicates whether this SetFont message is simply an update message or whether it is a "real" SetFont message meaning that this particular view should set its own font to the new font.

```
virtual void SetSizing(BOOLEAN isSizable);
```

Takes a Boolean value that sets the sizing of a text object to TRUE so that this object can be sized or to FALSE so that it cannot be sized. This method overrides the inherited in order to disable text sizing.

```
virtual void Size(const CRect& theRect);
```

Sizes the text box according to the dimensions of theRect. Currently, this method has no effect since text can only be sized by changing its font.

Protected Methods

None.

Private Methods

```
void RecalculateSize(void);
```

When the size, the position, or the font of the text object changes, this internal method is called to recalculate its size.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

```
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
```

```
virtual void Deactivate(void);
```

```
virtual void Disable(void);
```

```
virtual void DoActivate(void);
```

```
virtual void DoCommand(long theCommand, void* theData=NULL);
```

```
virtual void DoDeactivate(void);
```

```
virtual void DoDisable(void);
```

```
virtual void DoDraw(void);
```

```
virtual void DoDraw(const CRect& theClippingRegion);
```

```
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual void Enable(void);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
```



```
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(Long theCommand);
virtual void SetDoubleCommand(Long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

Overrides

You can override the selection routines with different selection actions.

Summary: CText.h

```
#ifndef CText_H
#define CText_H

#include "CSubview.h"

class CText : public CView
{
public:
    // Construct, init, destruct:
    CText(CSubview* theEnclosure, const CPoint& theTopLeft,
          const CString theText = NULLString);
    CText(const CText& theText);
    CText& operator=(const CText& theText);
    ~CText(void);

    BOOLEAN IText(const CString& theText,
                  BOOLEAN isOpaque = FALSE,
                  BOOLEAN isVisible = TRUE,
                  long theGlue = NULLSTICKY);

    // Setting and getting the text:
    virtual void SetText(const CString& theText);
    virtual void SetTitle(const CString& theText);
    virtual const CString GetText(void) const;

    virtual BOOLEAN IsOpaque(void) const;
    virtual void SetOpaque(BOOLEAN isOpaque);

    // Selecting text:
    virtual void Select(void);
    virtual void Deselect(void);
    virtual BOOLEAN IsSelected(void) const;

    // Size util:
    virtual UNITS GetHeight(void) const;
    virtual UNITS GetWidth(void) const;

    // Inherited utilities:
    virtual void Draw(const CRect& theClippingRegion);
    virtual void SetFont(const FONT& theFont,
                        BOOLEAN isUpdate = FALSE);
    virtual void SetSizing(BOOLEAN isSizable);
    virtual void Size(const CRect& theRect);

private:
    BOOLEAN itIsOpaque // Is the background transparent or opaque?
    BOOLEAN itIsSelected; // is the text selected?
    int itsAscent; // drawing distance from top

    void RecalculateSize(void);
};

#endif CText_H
```

CUnits

CString

CPoint

CRect

CUnits

Description

CUnits is a class that allows you to use logical units to define the coordinates on the screen. The use of logical units makes applications more portable because different machines have different screen widths, heights, and resolutions. The logical units are mapped out to the physical device on which you are displaying your information. We use the term “physical device” because the units can be mapped not only to a screen display but also to a printer display. If they are mapped to the screen, the logical units are translated into pixels; if they are mapped to a printer, then the logical units are translated into printer units.

Heritage

Superclass: none

Usage

By default, all XVT-Power++ applications use a one-to-one pixel mapping for drawing or printing. You can set a different mapping—in centimeters, inches, characters, or a user-defined unit—by instantiating a CUnits object and then calling a certain object in the application framework hierarchy and setting its units through its SetUnits method.

CUnits objects are treated just like CEnvironment objects in that objects inherit the units of their enclosures, unless they have their own private units. A view enclosure uses its own units, if it has any, or it inherits the units of its enclosure, which could be a window. A window can either have its own units or use the units of its document. Finally, there is a global CUnits object from which all objects in the application framework hierarchy can inherit.

Enums

Output Device

SCREEN	translate units to screen pixels during execution
PRINTER	translate units to printer pixels during execution

Public Data Members

extern	FONT STDFONT;	defined by CEnvironment
--------	---------------	-------------------------

Protected Data Members

None.

Private Data Members

static const CUnits*	itsGlobalUnits;	global units object
OutputDevice	itsOutputDevice;	

Mapping values, where: logical * mapping = physical

float	itsHMapping;	current horizontal mappings
float	itsVMapping;	current vertical mappings
float	itsHScreenMapping	horizontal screen mappings
float	itsVScreenMapping;	vertical screen mappings
float	itsHPrinterMapping	horizontal printer mappings
float	itsVPrinterMapping;	vertical printer mappings

Development Metrics

int	itsScreenWidth, itsScreenHeight;	
int	itsHResolution, itsVResolution;	
BOOLEAN	itIsMapping;	dynamic device-to-runtime map

Object ownership:

CBoss*	itsOwner;	the object owning the units
--------	-----------	-----------------------------

Macros

The following macros allow easy specification that a unit is physical. They should be used only within classes derived from CBoss. Each of these macros tests whether an object has any units. If it does, a macro takes the pixels and translates them to logical units.

```

#define HLogical(thePixels) (itsUnits? \
    itsUnits->HPhysicalToLogical(thePixels): \
    thePixels)
#define VLogical(thePixels) (itsUnits? \
    itsUnits->VPhysicalToLogical(thePixels): \
    thePixels)
#define HPhysical(thePixels) (itsUnits? \
    itsUnits->HLogicalToPhysical(thePixels): \
    int(thePixels))
#define VPhysical(thePixels) (itsUnits? \
    itsUnits->VLogicalToPhysical(thePixels): \
    int(thePixels))

```

Public Methods

```

CUnits(float theHScreenMapping = 1.0,
        float theVScreenMapping = 1.0,
        float theHPrinterMapping = 1.0,
        float theVPrinterMapping = 1.0,
        OutputDevice theDevice = SCREEN);

```

A constructor. It takes four floats that represent the horizontal and vertical screen mappings and the horizontal and vertical printer mappings. The default value of 1.0 for each mapping means that each logical unit maps 1.0 to a pixel. Each logical unit is multiplied by the mapping to get a physical unit. For example, if each logical unit is two pixels for the horizontal screen mapping, then the mapping would be 2. If each 10 units should equal 25 pixels, then 2.5 would be the horizontal screen mapping, and so on. You can have different mappings for the screen and the printer and different mappings for horizontal and vertical units.

```

void SetOwner(CBoss* theOwner);

```

Sets the owner of the units, which is important for updating purposes. When a certain unit's mappings change, the CUnits object automatically sends an UpdateUnits message to its owner, which is set through SetOwner. Once the owner is notified of the update, it updates itself and propagates the update message to any of its child objects. For example, the application would update all of its documents, the documents would update all of their windows, and so on, but only if they are sharing the same CUnits object. Basically, the owner of a CUnits object is the topmost object in the hierarchy that is using the CUnits object.

CBoss* GetOwner(void) const;

Returns the owner of the CUnits object, which is the topmost object in the hierarchy that is using the CUnits object. *See* SetOwner.

Output Device Methods

For the following methods, the screen is the default output device.

void SetOutputDevice(OutputDevice theDevice);

Sets the output device, which can be either the screen or the printer, for the CUnits object. The output device that is being used determines the mappings used to do the translation.

OutputDevice GetOutputDevice(void) const;

Returns the output device for the CUnits object.

Methods for Selecting Development-to-Execution Mappings

void SetDynamicMapping(BOOLEAN isMapping);

Allows you to turn the dynamic mapping on or off. When dynamic mapping is on, unit conversion is based on a comparison of the development metrics and the execution metrics. The development metrics are hard-coded into the program by calling SetDevelopmentMetrics before setting the dynamic mapping. The execution metrics are computed at run time.

BOOLEAN GetDynamicMapping(void) const;

Returns a Boolean value of TRUE if dynamic mapping is turned on or FALSE if it is turned off.

void SetDevelopmentMetrics(int theDeviceWidth, int theDeviceHeight, int theHResolution, int theVResolution);

Allows you to hard-code the metrics that you used in developing your application. These metrics are used for calculating the execution metrics when dynamic mapping is on. To find out what values should be hard-coded as the development metrics, you can use the get_value functions provided by the XVT Portability Toolkit. These functions return the values of the metrics.

Methods for Setting User-Defined Mappings, Where: Logical * Mapping = Physical

```
void SetScreenMapping(float theHScreenMapping,
    float theVScreenMapping);
```

Sets the horizontal and vertical screen mappings for the CUnits object.

```
void SetPrinterMapping(float
    theHPrinterMapping, float
    theVPrinterMapping);
```

Sets the horizontal and vertical printer mappings for the CUnits object.

```
float GetHScreenMapping(void) const;
```

Returns the horizontal screen mapping for the CUnits object.

```
float GetVScreenMapping(void) const;
```

Returns the vertical screen mapping for the CUnits object.

```
float GetHPrinterMapping(void) const;
```

Returns the horizontal printer mapping for the CUnits object.

```
float GetVPrinterMapping(void) const;
```

Returns the vertical printer mapping for the CUnits object.

Logical-To-Physical Conversion Methods

You can call the following methods when you want to translate a certain logical coordinate to a certain physical coordinate using the specified units.

```
int HLogicalToPhysical(UNITS theLogicalUnit)
    const;
```

Takes a logical unit and returns an integer. It uses the horizontal mappings according to the output device currently being used, and it maps the logical units to physical units. theLogicalUnit can take a UNITS value in centimeters, inches, characters, pixels, or a user-defined unit.

```
int VLogicalToPhysical(UNITS theLogicalUnit)
    const;
```

Takes a logical unit and returns an integer. It uses the vertical mappings according to the output device currently being used, and it maps the logical units to physical units. theLogicalUnit can take a UNITS value in centimeters, inches, characters, pixels, or a user-defined unit.

```
RCT LogicalToPhysical(UNITS theLeft, UNITS  
theTop, UNITS theRight, UNITS theBottom)  
const;
```

Takes a UNITS value for four different points (left, top, right, and bottom) and returns an XVT Portability Toolkit RCT.

```
PNT LogicalToPhysical(UNITS theHPos, UNITS  
theVPos) const;
```

Takes values for the horizontal and vertical positions of a coordinate and returns an XVT Portability Toolkit PNT.

Physical-To-Logical Conversion Methods

```
UNITS HPhysicalToLogical(int thePhysicalUnit)  
const;
```

Takes a physical unit for a horizontal mapping and returns its logical unit.

```
UNITS VPhysicalToLogical(int thePhysicalUnit)  
const;
```

Takes a physical unit for a vertical mapping and returns its logical unit.

```
CPoint PhysicalToLogical(const PNT thePnt)  
const;
```

Takes an XVT Portability Toolkit PNT and returns an XVT-Power++ CPoint equivalent.

```
CRect PhysicalToLogical(const RCT& theRct)  
const;
```

Takes an XVT Portability Toolkit RCT and returns an XVT-Power++ CRect equivalent.

Static Methods for Global Unit Object Settings

```
static void SetGlobalUnits(const CUnits*  
theUnits);
```

The CUnits class keeps track of a static global pointer to a certain CUnits object that has been defined as the global object. This global object is used to compute the current units to be used for translation. SetGlobalUnits sets this global pointer. Usually units are translated automatically when it is time to call an XVT Portability Toolkit drawing function. Any XVT-Power++ CRect or CPoint object is assumed to be in logical units. As soon a CRect or a CPoint is translated into an XVT Portability Toolkit RCT or PNT, it is converted to physical units as specified by the global CUnits object. Internally, XVT-Power++ takes care of setting the global units when

necessary so that the units are converted automatically when different objects are drawn.

```
static const CUnits* GetGlobalUnits(void)  
const;
```

Returns a pointer to the object that is currently defined as the global CUnits object. See SetGlobalUnits.

Protected Methods

```
void UpdateOwner(void);
```

A method that is called internally to update the owner of a CUnits object whenever the units change. When a certain unit's mappings change, the CUnits object automatically sends an *update units* message to its owner, which is set through SetOwner. Once the owner is notified of the update, it updates itself and propagates the update message to any of its child objects. For example, the application would update all of its documents, the documents would update all of their windows, and so on, but only if they are sharing the same CUnits object. Basically, the owner of a CUnits object is the topmost object in the hierarchy that is using the CUnits object.

Private Methods

None.

Classes Derived From CUnits: CCharacterUnits

Description

Given a font, CCharacterUnits calculates the mappings for the average height and width of a character in that font. This average becomes the standard unit. If you specify that you want an object to measure three units horizontally, its width is approximately the width of three characters.

Data Members (Private)

```
FONT                    itsFont;
```

Public Methods

```
CCharacterUnits(const FONT& theBaseFont =  
STDFONT);
```

A constructor. You must specify a font type (Times, Helvetica, Courier, and so on) as the base font.

```
CCharacterUnits(const CWindow* theFontWindow,  
int theFontFamily,int theStyle, short  
theSize);
```

A constructor. It takes a pointer to the window in which the font will appear and integers specifying the font family (Times, Helvetica, Courier, and so on), style (italics, bold, and so on), and size (in points).

```
void SetBaseFont(const CWindow* theFontWindow,  
const FONT& theBaseFont = STDFONT);
```

Given a pointer to a window and a base font, this method sets this font to be the window's base font.

```
void SetBaseFont( int theFontFamily, int  
theStyle, short theSize);
```

Given integers for a font family (Times, Helvetica, Courier, and so on), style (italics, bold, and so on), and size (in points), this method sets the described font to be the base font of the CCharacterUnits object.

```
const FONT* GetBaseFont(void) const;
```

Returns a pointer to the font that is the base font of the CCharacterUnits object.

```
static void NeedsPrinterMapping(BOOLEAN  
needsMapping);
```

Takes a Boolean value of TRUE if your system needs printer font definition files and FALSE if it does not.

Private Methods

```
void GetCharSize(WINDOW theXVTWindow, int*  
theWidth, int* theHeight);
```

An internal method that returns the width and height of a character in the designated the XVT Portability Toolkit window.

Classes Derived From CUnits: CinchUnits

Description

CInchUnits calculates all the horizontal and vertical screen/printer mappings so that each unit is an inch.

Public Methods

```
CInchUnits(void);
```

A constructor.

Classes Derived From CUnits: CCentimeterUnits

Description

CCentimeterUnits calculates all the horizontal and vertical screen/printer mappings so that each unit is a centimeter.

Public Methods

CCentimeterUnits(void);

A constructor.

Summary: CUnits.h

```
#ifndef CUnits_H
#define CUnits_H

#include "PwrNames.h"
#include "PwrDef.h"
#include "CGlobal.h"
#include "xvt.h"

class CRect;
class CPoint;
class CUnits;
class CWindow;
class CBoss;

extern FONT STDFONT; // Defined by CEnvironment;

typedef enum
{
    SCREEN, // Translate units to screen pixels during execution
    PRINTER // Translate units to printer pixels during execution
}
OutputDevice;

class CUnits
{
public:
    CUnits(float theHScreenMapping = 1.0,
           float theVScreenMapping = 1.0,
           float theHPrinterMapping = 1.0,
           float theVPrinterMapping = 1.0,
           OutputDevice theDevice = SCREEN);

    void SetOwner(CBoss* theOwner);
    CBoss* GetOwner(void) const;

    // Output device: SCREEN is used by default.

    void SetOutputDevice(OutputDevice theDevice);
    OutputDevice GetOutputDevice(void) const;

    // Methods for selecting development-to-execution mappings:
    void SetDynamicMapping(BOOLEAN isMapping);
    BOOLEAN GetDynamicMapping(void) const;

    void SetDevelopmentMetrics(int theDeviceWidth, int theDeviceHeight,
                              int theHResolution, int theVResolution);
```

```

// Methods for setting user defined mappings, where:
// logical * mapping = physical

void SetScreenMapping(float theHScreenMapping,
                     float theVScreenMapping);

void SetPrinterMapping(float theHPrinterMapping,
                      float theVPrinterMapping);

float GetHScreenMapping(void) const;
float GetVScreenMapping(void) const;

float GetHPrinterMapping(void) const;
float GetVPrinterMapping(void) const;

// Logical-To-Physical conversion methods:

int HLogicalToPhysical(UNITS theLogicalUnit) const;
int VLogicalToPhysical(UNITS theLogicalUnit) const;

RCT LogicalToPhysical(UNITS theLeft, UNITS theTop,
                     UNITS theRight, UNITS theBottom) const;

PNT LogicalToPhysical(UNITS theHPos, UNITS theVPos) const;

// Physical-To-Logical conversion methods:
UNITS HPhysicalToLogical(int thePhysicalUnit) const;
UNITS VPhysicalToLogical(int thePhysicalUnit) const;

CPoint PhysicalToLogical(const PNT thePnt) const;
CRect PhysicalToLogical(const RCT& theRct) const;

// Static Methods for global unit object settings:
static void SetGlobalUnits(const CUnits* theUnits);
static const CUnits* GetGlobalUnits(void) const;

protected:
    void UpdateOwner(void);

private:
    static const CUnits* itsGlobalUnits; // Global units object
    OutputDevice itsOutputDevice;

    // Mapping values, where: logical * mapping = physical
    float itsHMapping; // Current hor. mappings
    float itsVMapping; // Current ver. mappings

    float itsHScreenMapping, itsVScreenMapping; // Screen mappings
    float itsHPrinterMapping, itsVPrinterMapping; // Printer mappings

    // Development metrics:
    int itsScreenWidth, itsScreenHeight;
    int itsHResolution, itsVResolution;

    BOOLEAN itIsMapping; // Dynamic dev-to-runtime map
    // Object ownership:
    CBoss* itsOwner; // The object owning the units
};

class CCharacterUnits : public CUnits
{
public:
    CCharacterUnits(const CWindow* theFontWindow,
                   const FONT& theBaseFont = STDFONT);

```

```

CCharacterUnits(const CWindow* theFontWindow, int theFontFamily,
               int theStyle, short theSize);

void SetBaseFont(const CWindow* theFontWindow,
               const FONT& theBaseFont = STDFONT);

void SetBaseFont(const CWindow* theFontWindow,
               int theFontFamily,
               int theStyle,
               short theSize);

const FONT* GetBaseFont(void) const;

private:
    FONT itsFont;

    void GetCharSize(WINDOW theXVTWindow, int* theWidth,
                   int* theHeight);
};

class CInchUnits : public CUnits
{
public:
    CInchUnits(void);
};

class CCentimeterUnits : public CUnits
{
public:
    CCentimeterUnits(void);
};

// The following macros allow easy specification that a unit is physical.
// They should only be used within CBoss derived classes.

#define HLogical(thePixels) (itsUnits? \
                           itsUnits->HPhysicalToLogical(thePixels) : \
                           thePixels)

#define VLogical(thePixels) (itsUnits? \
                           itsUnits->VPhysicalToLogical(thePixels) : \
                           thePixels)

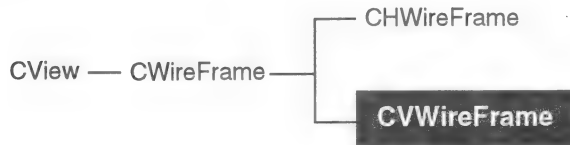
#define HPhysical(thePixels) (itsUnits? \
                           itsUnits->HLogicalToPhysical(thePixels) : \
                           int(thePixels))

#define VPhysical(thePixels) (itsUnits? \
                           itsUnits->VLogicalToPhysical(thePixels) : \
                           int(thePixels))

#endif CUnits_H

```

CVWireFrame



Description

CVWireFrame is a class that overrides CWireFrame in order to implement special wire frame behavior: vertical-only dragging. This behavior is achieved by trapping all `MouseMove` events and fixing the horizontal position to a constant value.

Heritage

Superclass: CWireFrame

Usage

Give a view a vertical wireframe as follows:

```
aView->SetWireFrame(new CVerticalWireFrame(aView));
aView->SetDragging(TRUE);
```

Data Members

None.

Public Methods

```
CVerticalWireFrame(CView *theOwner, CSubview*  
theGroupEnclosure = NULL);
```

Calls the inherited constructor, that is, the constructor of CWireFrame. `theOwner` is a pointer to its owner, the CView object with which it is associated. It gets information about this object, such as the object's enclosure, from the object itself. `theGroupEnclosure` is the subview that acts as the enclosure of the group containing this wire frame.

```
virtual void MouseMove(CPoint theLocation, short  
theButton = 0, BOOLEAN isShiftKey=FALSE,  
BOOLEAN isControlKey=FALSE);
```

An overridden method that suppresses horizontal movement. When the user moves the mouse over a view containing local coordinate `theLocation`, the view receives and handles this event. If the wire frame is in a selected state, it responds to the

mouse move event by resizing or dragging the rubberband frame in a vertical direction. theButton specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right). By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoActivate(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,   BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
```

```
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsVisible(void) const;
```



```

virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CWireFrame

```

virtual void ClearGroupRegion(void);
virtual void Deselect(void);
virtual void DoAutoScroll(int theHorizontalIncrement, int theVerticalIncrement);
virtual BOOLEAN DoBorderCheck(const CPoint& theLocation);
virtual void DoDeselect(CView* theView);
virtual BOOLEAN DoDraggingCheck(const CPoint& theLocation);
virtual void Drag(CPoint theNewLocation);
virtual void Draw(const CRect& theClippingRegion);
virtual void DrawFrameGrabbers(DRAWSTATE isDrawing);
virtual void DrawWireFrame(DRAWSTATE isDrawing);
virtual CSubview* GetGroupEnclosure(void) const;
virtual CView* GetOwner(void) const;
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSelected(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual void MouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

```

```

virtual void ReSize(CPoint theNewLocation);
virtual void Select(void);
virtual void SetDragging(BOOLEAN isDraggable);
void SetGroupEnclosure(CSubview* theGroupEnclosure);
void SetGroupRegion(void);
virtual void SetFont(const FONT& theFont, BOOLEAN isUpdate = FALSE);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void Size(const CRect& theNewSize);
BOOLEAN SizeOwner(void);

```

Summary: CVWireFrame.h

```

#ifndef CVWireFrame_H
#define CVWireFrame_H

#include "PwrDef.h"
#include CWireFrame_i

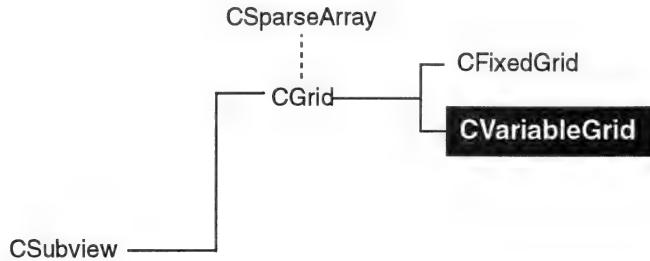
class CVerticalWireFrame : public CWireFrame
{
public:
    CVerticalWireFrame(CView *theOwner, CSubview* theGroupEnclosure = NULL) :
        CWireFrame(theOwner, theGroupEnclosure)
    {
    }

    virtual void MouseMove(CPoint theLocation, short theButton = 0,
        BOOLEAN isShiftKey=FALSE,
        BOOLEAN isControlKey=FALSE)
    {
        // By using itsGrabPoint.H(), we pretend that all horizontal positions
        // are identical to the original "grab" position:
        theLocation.H(theGrabPoint.H());
        CWireFrame::MouseMove(theLocation, theButton, isShiftKey,
            isControlKey);
    }
};

#endif

```

CVariableGrid



Description

A CVariableGrid is a grid with variable-sized rows and columns; each row or column can have its own different size. A spreadsheet is an example of the use of a variable grid; each of the columns has a different width. For more information, see CGrid and CFixedGrid.

Heritage

Superclass: CGrid

Subclasses: None

Usage

Create a grid, initialize it, and insert objects into its cells. Several methods are provided to manipulate the cell sizes.

Environment

The lines of a variable grid are drawn with the *pen*, and the area within each cell is painted with the *brush*. You can set the color and pattern of both the pen and the brush. Also, you can set the pen width.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

<code>CIdOrderedList</code>	<code>*itsRowSizes;</code>	sparse list of widths
<code>CIdOrderedList</code>	<code>*itsColumnSizes;</code>	sparse list of heights

Public Methods

Constructor, Destructor, and Initializer Methods

```
CVariableGrid(CSubview* theEnclosure, const  
CRect& theRegion, int theNumberOfRows, int  
theNumberOfColumns);
```

A constructor. `theEnclosure` is a pointer to the subview that will contain the grid. `theRegion` is a coordinate location, local to the enclosure, that is used to place the grid. This method also requires you to specify the number of rows and columns in the grid, and it divides the given region into rows and columns.

```
CVariableGrid(CSubview* theEnclosure,  
CPoint& theTopLeftCorner,  
UNITS theDefaultRowHeight,  
UNITS theDefaultColumnWidth,  
int theNumberOfRows,  
int theNumberOfColumns);
```

A constructor, which takes an enclosure for the grid and requires you to specify the number of rows and columns. `theTopLeftCorner` is a `CPoint` coordinate for placing the top-left corner of the grid. You specify the size of the grid by giving it a default row height and a default column width. By default, a `CVariableGrid` is identical to a `CFixedGrid`; that is, all of the cells have a default size that is set at creation time. However, you can vary the size of individual rows and columns as much as you want.

```
CVariableGrid(const CVariableGrid& theGrid);
```

A copy constructor that duplicates the attributes, but not the contents, of a grid. That is, it copies the grid but not the objects it contains.

```
CVariableGrid& operator=(const CVariableGrid&  
theGrid);
```

An assignment operator that duplicates the attributes, but not the contents, of a grid. That is, it copies the grid but not the objects it contains.

```
virtual ~CVariableGrid(void);
```

The destructor. It cleans up after the grid and deletes any subviews it contains.

```
BOOLEAN IVariableGrid(BOOLEAN isClipping = TRUE,  
                       PLACEMENT thePlacement = TOPLEFT,  
                       BOOLEAN isGridVisible = FALSE,  
                       POLICY theSizingPolicy = ADJUSTCellSize,  
                       BOOLEAN isVisible = TRUE,  
                       GLUETYPE theGlue = NULLSTICKY);
```

The initializer. The `isClipping` parameter takes a Boolean value indicating whether the items placed in the grid are clipped to their cells. By default, clipping is turned off. `thePlacement` specifies the orientation for placing items in their cells—in the top left corner by default. `isGridVisible` sets whether the lines of the grid are visible or not. `theSizingPolicy` takes a value indicating one of the two different sizing policies for grids: `ADJUSTCellSize` (the default) or `ADJUSTCellNumber`. The `isVisible` parameter sets the visibility state for the entire grid object (as opposed to `isGridVisible`, which pertains to the horizontal and vertical lines within the grid). Finally, this initializer takes a glue type.

Cell Operations Inherited from CGrid

```
virtual CRect GetCellSize(int theRow, int  
                          theColumn) const;
```

Gets the size of the cell located at the given row and column. It returns a `CRect`, consisting of that cell's coordinates, which are local to the grid.

```
virtual int GetRow(UNITS theHorizontalLocation)  
                const;
```

Gets a row of the grid. It takes a horizontal location in coordinates relative to the grid and returns the number of the row that contains the given point.

```
virtual int GetCol(UNITS theVerticalLocation)  
                const;
```

Gets a column of the grid. It takes a vertical location in coordinates relative to the grid. It returns the number of the column that contains `theVerticalLocation`.

```
virtual UNITS GetWidth(int theColumn) const;
```

Returns the width, in pixels, of the specified column, beginning at the left with zero (0).

```
virtual UNITS GetHeight(int theRow) const;
```

Returns the height, in pixels, of the specified row, beginning at the top with zero (0).

Inherited Sizing Method

```
virtual void Size(const CRect& theNewSize);
```

Sizes the grid according to the coordinates of theNewSize. Sizing of a grid varies according to the sizing policy that has been set. This method overrides the corresponding inherited method on CGrid.

Sizing Methods

```
virtual void SizeCol(int theColumn, UNITS  
theNewWidth);
```

Sizes a column of the grid, taking the column number and the new width. If this column was previously using a default width, it will no longer use it.

```
virtual void SizeRow(int theRow, UNITS  
theNewHeight);
```

Sizes a row of the grid, taking the row number and the new height. If this row was previously using a default height, it will no longer use it.

```
virtual void SetDefaultWidth(UNITS  
theNewDefaultWidth);
```

Resets the default width for the rows of the grid that are using the default width because no special widths have been set for them.

```
virtual void SetDefaultHeight(UNITS  
theNewDefaultHeight);
```

Resets the default height for the columns of the grid that are using the default height because no special heights have been set for them.

```
virtual void SizeCell(UNITS newDefaultWidth,  
UNITS newDefaultHeight);
```

Resizes the cells of the grid that are using the default size because no special widths and heights have been set for them. This method takes a new default width and a new default height and allows you to adjust the grid's defaults.

```
virtual void SetNumCells(int theNumberOfRows,  
int theNumberOfCols);
```

Enlarges or reduces the size of the entire grid by giving it the specified number of rows and number of columns. This method overrides the corresponding inherited method on CGrid.

```
virtual void AdjustRow(int theRow, ADJUST  
theAdjustment);
```

Takes the given row and an ADJUST parameter that has a value of either MAXIMIZE or MINIMIZE. This method goes through all of the objects contained in the cells of the row, finds either the largest or the smallest object (depending on the value of ADJUST), and resizes the row to fit the height of that object. This method is very similar to the AdjustCells method on the CGrid class, except that is specific to a row.

```
virtual void AdjustCol(int theCol, ADJUST  
theAdjustment);
```

Takes the given column and an ADJUST parameter that has a value of either MAXIMIZE or MINIMIZE. This method goes through all of the objects contained in the cells of the column, finds either the largest or the smallest object (depending on the value of ADJUST), and resizes the column to fit the width of that object. This method is very similar to the AdjustCells method on the CGrid class, except that is specific to a column.

Private Methods

```
void ResetWidth(void);
```

An internal method that resets the default width for the grid after some rows are resized or moved.

```
void ResetHeight(void);
```

An internal method that resets the default height for the grid after some columns are resized or moved.

```
void ResetFrame(void);
```

An internal method that resets the size of the entire grid after it has been sized or moved.

```
int GetRowNumber(UNITS theNewHeight);
```

Provides an internal utility that, given a horizontal location, returns the number of the row into which that location fits. Some special calculations must be done internally because variable grids can have variable-sized rows.

```
int GetColNumber(UNITS theNewWidth);
```

Provides an internal utility that, given a vertical location, returns the number of the column into which that location fits. Some special calculations must be done internally because variable grids can have variable-sized columns.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CWindow* GetCWindow(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
```



```

virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CSubview

```

virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);

```

```

virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation,CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

From CGrid

```

virtual void AdjustCells(ADJUST theAdjustment);

```

```
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoPlaceView(void);
virtual void DoSize(const CRect& theNewSize);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* GetContents(CPoint theLocation) const;
virtual CView* GetContents(int theRow, int theCol) const;
CView* GetHitItem(CPoint theLocation) const;
virtual int GetNumCols(void) const;
virtual int GetNumRows(void) const;
virtual PLACEMENT GetPlacement(void) const;
virtual void GetPosition(const CView* theView, int *theRow, int *theCol) const;
virtual void GetPosition(const CRect& theView, int *theRow, int *theCol) const;
virtual POLICY GetSizingPolicy(void) const;
virtual void HideGrid(void);
virtual void Insert(CView* theView);
virtual void Insert(CView* theView, int theRow, int theColumn);
virtual BOOLEAN IsClipping(void);
virtual BOOLEAN IsGridVisible(void);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void PlaceView(CView* theView, int theRow, int theCol);
virtual CView* Remove(int theRow, int theColumn);
virtual BOOLEAN Remove(const CView* theView);
void RemoveSubview(const CView* theView);
virtual CView* Replace(CView* theView);
virtual CView* Replace(CView* theView, int theRow, int theColumn);
void ResetCell(CView* theView, int theNewRow=-1, int theNewCol=-1);
virtual void SetClipping(BOOLEAN isClipping);
```

```
void SetDragPoint(const CPoint& theLocation);
virtual void SetPlacement(PLACEMENT thePlacement);
virtual void SetSizingPolicy(POLICY thePolicy);
virtual void ShowGrid(void);
virtual BOOLEAN Validate(int theRow, int theCol);
```

Summary: CVariableGrid.h

```
#ifndef CVariableGrid_H
#define CVariableGrid_H

#include "CGrid.h"

class COrderedList;

class CVariableGrid : public CGrid
{
public:
    // Create, Initialize, Destruct:
    CVariableGrid(CSubview* theEnclosure, const CRect& theRegion,
        int theNumberOfRows, int theNumberOfColumns);

    CVariableGrid(CSubview* theEnclosure,
        CPoint& theTopLeftCorner,
        UNITS theDefaultRowHeight,
        UNITS theDefaultColumnWidth,
        int theNumberOfRows,
        int theNumberOfColumns);

    CVariableGrid(const CVariableGrid& theGrid);
    CVariableGrid& operator=(const CVariableGrid& theGrid);
    virtual ~CVariableGrid(void);

    BOOLEAN IVariableGrid(BOOLEAN isClipping      = TRUE,
        PLACEMENT thePlacement                    = TOPLEFT,
        BOOLEAN isGridVisible                     = FALSE,
        POLICY theSizingPolicy                    = ADJUSTCellSize,
        BOOLEAN isVisible                         = TRUE,
        GLUETYPE theGlue                         = NULLSTICKY);

    //Cell Operations
    virtual CRect GetCellSize(int theRow, int theColumn) const;
    virtual int GetRow(UNITS theHorizontalLocation) const;
    virtual int GetCol(UNITS theVerticalLocation) const;

    virtual UNITS GetWidth(int theColumn) const;
    virtual UNITS GetHeight(int theRow) const;

    // Sizing:
    virtual void Size(const CRect& theClippingRegion);
    virtual void SizeCol(int theColumn, UNITS theNewWidth);
    virtual void SizeRow(int theRow, UNITS theNewHeight);

    virtual void SetDefaultWidth(UNITS theNewDefaultWidth);
    virtual void SetDefaultHeight(UNITS theNewDefaultHeight);

    virtual void SizeCell(UNITS newDefaultWidth, UNITS newDefaultHeight);
    virtual void SetNumCells(int theNumberOfRows, int theNumberOfCols);

    virtual void AdjustRow(int theRow, ADJUST theAdjustment);
    virtual void AdjustCol(int theCol, ADJUST theAdjustment);
```

```
protected:
```

```
private:
```

```
    CIdOrderedList *itsRowSizes; // sparse list of widths  
    CIdOrderedList *itsColumnSizes; // sparse list of heights
```

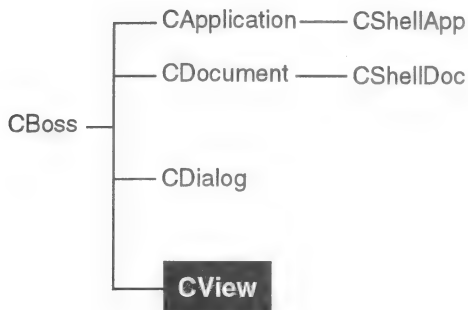
```
    void ResetWidth(void);  
    void ResetHeight(void);  
    void ResetFrame(void);
```

```
    int GetRowNumber(UNITS theNewHeight);  
    int GetColNumber(UNITS theNewWidth);
```

```
};
```

```
#endif CVariableGrid_H
```

CView



Description

A *view* is any object that can be displayed on the screen in a window or inside other views. Views can receive events, commands, and environment changes, and they can become visible/invisible or active/inactive. View objects are instantiated, initialized, and automatically registered with a window or another view. After that, these objects take care of updating, refreshing, handling mouse events, sizing, moving, clipping to enclosures, scrolling, and so on.

All objects in the CView hierarchy can be enabled or disabled, meaning that they can receive events or not receive events. However, even when a view is disabled, it can still receive moving and sizing events. Setting the moving and sizing to TRUE in effect disables a view because that view no longer acts as it usually does—as a button, for example. Instead, it simply acts as an object that is being moved and sized.

One special kind of CView, the subview, has the added ability to contain other views recursively nested within it. To manage the subviews, XVT-Power++ provides different types of methods. One type, the “Do” methods, are message passing functions that propagate events to recursively nested subviews. For example, DoShow propagates the Show message, notifying each nested subview to reveal itself and to propagate the message recursively on down to its nested subviews. This capability for propagation is very important for classes derived from CSubview because subviews often need to propagate a message to all of their nested views.

CView is a more general case because it does not nest subviews. XVT-Power++ is structured so that all viewable objects have the

same basic functionality, whether they are a CView or a CSubview. That is, there is a “wide interface” to CView and its derived classes. You do not need to know whether a particular view has subviews. You simply call the “Do” method, and that method propagates through all the views. If the view cannot contain subviews, then it just propagates the message to itself. Every time there is a “Do” method at the CView level, this method has been provided for a “wide interface” so that CView objects can be handled just like CSubview objects, as is often noted in the following discussion of the methods on CView.

Heritage

Superclass: CBoss

Subclasses: CSubview, CNativeTextEdit, CText, CWireFrame, CWindow

Usage

This is an abstract class from which you must derive further classes. For more information, see the derived class CSubview.

Public Data Members

None.

Protected Data Members

CRect	itsFrame;	a view's rectangular border
CPoint	itsOrigin;	the view's origin for scrolling
CWindow	*itsCWindow;	the window in which it belongs
long	itsCommand;	command to generate upon a mouse click
long	itsDoubleCommand;	command to generate upon a double mouse click
CSubview	*itsEnclosure;	the enclosure to which the object belongs

Private Data Members

CEnvironment	*itsEnvironment;	its local environment
BOOLEAN	itIsVisible;	whether it is visible within the frame
BOOLEAN	itIsEnabled;	whether it can receive events
BOOLEAN	itIsActive;	whether it is in an

CString	itsTitle;	active state
int	itsId;	the view's title
CGlue	*itsGlue;	the view's ID number
CWireFrame	*itsWireFrame;	the type of glue
		the wire frame used for
		sizing and dragging
static CView*	itsClickedView;	used to generate click
		commands

Public Methods

Constructor and Destructor Methods

All of the CView constructors are called by the constructors of the classes derived from CView.

CView(CSubview* theEnclosure, const CRect& theRegion);

A constructor. theEnclosure is a pointer to the subview that will contain the view. theRegion is a coordinate location, local to the enclosure, that is used to place the view.

CView(const CView& theView);

A copy constructor, which takes another view. It takes the attributes of the view it is copying and mimics them.

CView& operator=(const CView& theView);

An assignment operator that is used for cloning a view.

virtual ~CView(void);

The destructor. It cleans up after the view is closed.

Visibility State Methods

virtual BOOLEAN IsVisible(void) const;

Takes no arguments but returns a Boolean value, which is TRUE if the view is visible and FALSE if the view is invisible.

virtual void Show(void);

Makes the view visible.

virtual void Hide(void);

Makes the view invisible.

virtual void DoShow(void);

Sends the view a message to become visible; the view reveals itself and in turn sends a message to its subviews to reveal themselves. DoShow is the "wide interface" counterpart of Show.

At the CView level, DoShow translates to a simple Show. However, it is overridden by CSubview to implement message passing to all of the subviews.

virtual void DoHide(void);

Sends the view a message to become invisible; the view hides itself and sends a message to its subviews to hide themselves. This method is the “wide interface” counterpart of Hide.

Activate Events

virtual BOOLEAN IsActive(void) const;

Indicates whether the state of a view is active or inactive. It returns a Boolean value of TRUE or FALSE.

virtual void Activate(void);

Sets the state of the view to active so that it is currently the view that will receive user input through keyboard or mouse events.

virtual void Deactivate(void);

Sets the state of the view to inactive, which means that it is currently not the view that will receive user input through keyboard or mouse events.

virtual void DoActivate(void);

Changes a view's state to active so that it is currently the view that will receive user input through keyboard or mouse events. The view in turn notifies its subviews to become active. In this case, the message translates as an Activate message.

virtual void DoDeactivate(void);

Changes a view's state so that it is currently not the view that will receive user input through keyboard or mouse events. The view in turn notifies its subviews to become inactive. In this case, the message translates to a Deactivate message.

Enable/Disable Events

virtual BOOLEAN IsEnabled(void) const;

Indicates whether a view is enabled to receive events (TRUE) or disabled (FALSE) so that it cannot receive events.

virtual void Disable(void);

Disables a view so that it cannot receive events.

virtual void Enable(void);

Enables a view so that it can receive events.

```
virtual void DoEnable(void);
```

Enables a view so that it can receive events. The view in turn notifies its subviews to become enabled. This message may be passed down until it reaches the deepest subview.

```
virtual void DoDisable(void);
```

Disables a view so that it cannot receive events. The view in turn notifies its subviews to become enabled. This message may be passed down until it reaches the deepest subview.

Mouse Events

Mouse events are passed directly to the view at the location where the event occurs. By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button. The base mouse methods have an argument called `theLocation`, which is the location of the cursor when the mouse button was clicked. This point, `theLocation`, is in *local* coordinates, that is, coordinates *relative* to the view's top-left corner. Each method also has a `theButton` argument that specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right). Typically, if you want to override the default parameters of a method for mouse events, you override the base method, not the "Do" method. For example, you would override the parameters of `MouseDown` rather than those of `DoMouseDown`.

Like the base mouse methods, the "Do" mouse methods take a parameter named `theLocation`. However, for the "Do" methods `theLocation` is in *global*, window-relative coordinates. The "Do" methods take the global coordinate, translate it into a local coordinate, and call the corresponding base mouse method. However, at the `CSubview` level, more computation is required to verify whether the global coordinate is contained in one of the subviews.

```
virtual void MouseDown(CPoint theLocation,  
                      short theButton      = 0,  
                      BOOLEAN isShiftKey   = FALSE,  
                      BOOLEAN isControlKey = FALSE);
```

When the user presses down a mouse button over a view containing local coordinate `theLocation`, the view receives and handles this event.

```
virtual void MouseMove(CPoint theLocation,  
                      short theButton      = 0,
```

```

        BOOLEAN isShiftKey  =FALSE,
        BOOLEAN isControlKey=FALSE);

```

When the user moves the mouse over a view containing local coordinate `theLocation`, the view receives and handles this event, whether the mouse button is down or not.

```

virtual void MouseUp(CPoint theLocation,
                    short theButton    = 0,
                    BOOLEAN isShiftKey  =FALSE,
                    BOOLEAN isControlKey=FALSE);

```

When the user releases a mouse button over a view containing local coordinate `theLocation`, the view receives and handles this event. This event is not necessarily preceded by a `MouseDown` or `MouseMove` event because those button actions may have taken place outside the view.

```

virtual void MouseDouble(CPoint theLocation,
                        short theButton    = 0,
                        BOOLEAN isShiftKey  = FALSE,
                        BOOLEAN isControlKey= FALSE);

```

When the user double clicks a mouse button over a view containing local coordinate `theLocation`, the view receives and handles this event.

```

virtual void DoMouseDouble(CPoint theLocation,
                          short theButton    = 0,
                          BOOLEAN isShiftKey  = FALSE,
                          BOOLEAN isControlKey= FALSE);

```

When the user double clicks the mouse at global coordinate `theLocation`, the view translates this coordinate into a local coordinate and calls `MouseDouble`.

```

virtual void DoMouseDown(CPoint theLocation,
                        short theButton    = 0,
                        BOOLEAN isShiftKey  = FALSE,
                        BOOLEAN isControlKey= FALSE);

```

When the user presses down a mouse button at global coordinate `theLocation`, the view translates this coordinate into a local coordinate and calls `MouseDown`.

```

virtual void DoMouseMove(CPoint theLocation,
                        short theButton    = 0,
                        BOOLEAN isShiftKey  = FALSE,
                        BOOLEAN isControlKey= FALSE);

```

When the user moves the mouse at global coordinate `theLocation`, the view translates this coordinate into a local coordinate and calls `MouseMove`.

```

virtual void DoMouseUp(CPoint theLocation,
                      short theButton    = 0,

```

```

    BOOLEAN isShiftKey = FALSE,
    BOOLEAN isControlKey= FALSE);

```

When the user releases a mouse button at global coordinate `theLocation`, the view translates this coordinate into a local coordinate and calls `MouseUp`.

```

virtual CView* FindHitView(const CPoint&
    theLocation) const;

```

A method that returns either a pointer to the view through which the method is invoked or a pointer to the view's wire frame in case the view has an active wire frame for moving and sizing (see `CWireFrame`). Whenever a view is about to receive a mouse event, `FindHitView` is called to find out whether the event should go to the view or to its wire frame.

Draw Events

```

virtual void Draw(const CRect&
    theClippingRegion);

```

Takes care of any drawing the view must do. `theClippingRegion` is the part of the view that needs to be drawn. If it is set to `NULL`, the entire view is drawn. `theClippingRegion` is in global, window-relative coordinates.

```

virtual void DoDraw(const CRect&
    theClippingRegion);

```

The view refreshes (draws) itself and notifies each of its subviews to do the same. `*theClippingRegion` is the part of the view that needs to be refreshed, and it is represented in global, window-relative coordinates; if it is set to `NULL`, the entire view is drawn. This method is automatically called after an `E_UPDATE` event.

```

virtual void Draw(void);

```

By default the clipping region is set to the entire region that the view occupies. If you do not want to calculate a clipping region and just want to draw the entire view, then you can call this method without passing anything.

```

virtual void DoDraw(void);

```

The view draws itself and notifies each of its subviews to do the same. By default the clipping region is set to the entire region that the view occupies. If you do not want to calculate a clipping region and just want to draw the entire view, then you can call this method without passing anything.

Other Events

```
virtual void Key(int theKey, BOOLEAN  
isShiftKey, BOOLEAN isControlKey);
```

Sends a keyboard event to a window. `int` is the ASCII value for the character key that was pressed; `isShiftKey` and `isControlKey` indicate whether the SHIFT or CONTROL key was pressed in conjunction with the character key.

```
virtual void DoKey(int theKey,BOOLEAN  
isShiftKey,BOOLEAN isControlKey);
```

A method that is called when there is a keyboard event. When the user presses a key on the keyboard, the XVT Portability Toolkit sends an `E_CHAR` event to the switchboard, which passes this event to the selected window. The window checks to see if one of its subviews is selected, and, if so, passes the event to the appropriate subview. If no subview is selected, the keyboard event propagates upwards from the window to the document and finally to the application. A keyboard event goes directly to the application if no windows are open.

`theKey` is the ASCII number of the character key that was pressed; `isShiftKey` and `isControlKey` indicate whether the SHIFT key or CONTROL key was pressed in conjunction with the character key. You can override `DoKey` to do something specific to a keyboard action within the application. The default mechanism does nothing.

```
virtual void Size(const CRect& theNewSize);
```

Resets the region of the view. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, `theNewSize`, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated.

```
virtual void DoSize(const CRect& theNewSize);
```

The “wide interface” counterpart of `Size`.

```
virtual void SetFont(const FONT& font, BOOLEAN  
isUpdate = FALSE);
```

Takes a font and sets the view’s environment to have that font. When the font of the global environment is changed, all views sharing that environment get a `SetFont` message with the `isUpdate` parameter set to `TRUE`. The `isUpdate` parameter indicates whether this `SetFont` message is simply an update message or whether it is a “real” `SetFont` message meaning that this particular view should set its own font to the new font.

```
virtual void DoSetFont(const FONT& theNewFont,  
    BOOLEAN isUpdate = FALSE);
```

The “wide interface” counterpart of SetFont.

Scrolling Methods

The following two methods are called when a scrollbar receives a scrolling event. The event propagates from the scrollbar to its enclosure. `theEventType` designates the kind of scroll: line up, line down, page up, or page down (the terms “up” and “down” are used for both horizontal and vertical scrolling, though the effects are different for horizontal scrolling). `theThumbPosition` is the numerical value for placement of the thumb when either thumb repositioning or dynamic thumb tracking is used. These methods are provided simply for “wide interface” purposes.

```
virtual void VScroll(SCROLL_CONTROL  
    theEventType, UNITS thePosition);
```

A method called when a scrollbar receives a vertical scroll event.

```
virtual void HScroll(SCROLL_CONTROL  
    theEventType, UNITS thePosition);
```

A method called when a scrollbar receives a horizontal scroll event.

Coordinates and Locations

CView has several convenient utility methods to get different coordinates that a view might have. These methods are used frequently. It is very important to know whether the coordinates of a given method are global, window-relative coordinates; local coordinates that are relative to the view’s enclosure; or local coordinates that are relative to the view itself.

```
virtual CRect GetFrame(void) const;
```

Returns the region of a view in coordinates that are relative to its enclosure.

```
virtual CRect GetGlobalFrame(void) const;
```

Returns the region of a view in global, or window-relative, coordinates.

```
virtual CRect GetClippedFrame(void) const;
```

Views are clipped to their enclosures, so portions of a view may not be visible. This method returns a region (CRect) with the coordinates of the visible portion of a clipped view. These coordinates are global, or window-relative. Usually, you will

call `GetClippedFrame` before you call `DoDraw` in order to pass the smallest region possible to `DoDraw`. Because `Draw` and `DoDraw` take window-relative coordinates, `GetClippedFrame` returns a window-relative `CRect`.

virtual `CRect GetLocalFrame(void) const;`

Returns the frame (`CRect`) in coordinates that are relative to the given view. For any view, `GetLocalFrame` returns a frame that has a top left corner of 0,0 and a bottom right corner equal to the view's width and height. Notice the difference between this method and `GetFrame`, which returns a region that is local to the enclosure. `GetLocalFrame` returns a region that is local to the given view specifically.

virtual `CPoint GetOrigin(void) const;`

Returns the view's origin (starting point), relative to its enclosing view.

virtual `CPoint GetGlobalOrigin(void) const;`

Returns the view's origin (starting point), relative to the window.

virtual void `SetOrigin(const CPoint& theDeltaPoint);`

Takes the current origin of the view and adds `theDeltaPoint` to it. For example, suppose a view has an origin of 10,20 and you want to shift the origin by three pixels to the right. You would set the origin by giving it a delta point of 3,0, which shifts the view three pixels horizontally.

virtual void `DoSetOrigin(const CPoint& theDeltaPoint);`

Takes the current origin of the view and adds `theDeltaPoint` to it. The view sends a message to each of its subviews to add `theDeltaPoint` to its current origin. This method is the "wide interface" counterpart of `SetOrigin`.

Environment Methods

Every view has an environment, which sets such variables as its font type, background and foreground colors, and pen width and pattern. See `CEnvironment`.

**virtual const `CEnvironment*`
`GetEnvironment(void) const;`**

For purposes of consistency, conciseness, and ease of use, views often share environments—with their enclosure, their window, their document, or their application. This method tests

to see if a view is using its own private environment. If not, it checks the environment of the view's enclosure. If the view's enclosure is not using an environment of its own, `GetEnvironment` recurses upward until it finds some environment that is being used and returns it. At the top level, the default environment of the `CApplication` object is always available.

```
virtual void SetEnvironment(const CEnvironment&
theNewEnvironment, BOOLEAN isUpdate =
FALSE);
```

Sets the environment for a view, using the `theNewEnvironment` that is passed to it. By default, views share their enclosure's environment. However, as soon as you use `SetEnvironment` to give a view an environment of its own, the view uses that environment instead of the shared environment.

When the global environment is changed, all views sharing that environment get a `SetEnvironment` message with the `isUpdate` parameter set to `TRUE`. The `isUpdate` parameter indicates whether this `SetEnvironment` message is simply an update message or whether it is a "real" `SetEnvironment` message meaning that this particular view should set its own environment to the new environment.

```
virtual void DoSetEnvironment(const
CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
```

Sets the environment for the view object, which notifies each of its subviews of the environment parameters of `theNewEnvironment`. This method is the "wide interface" counterpart of `SetEnvironment`.

When a view's environment is changed with `DoSetEnvironment`, all of its nested views get a `DoSetEnvironment` message with the `isUpdate` parameter set to `TRUE`. The `isUpdate` parameter indicates whether this `DoSetEnvironment` message is simply an update message or whether it is a "real" `DoSetEnvironment` message meaning that this particular view should set its own environment to the new environment and pass this change down to all of its child views that are sharing its environment.

```
virtual BOOLEAN IsEnvironmentShared(void);
```

A convenient method for finding out if a view is sharing an environment with another object. It returns `FALSE` if the view is using an environment of its own.

Glue Methods

virtual GLUETYPE GetGlue(void) const;

Returns the glue type of a view.

virtual void SetGlue(GLUETYPE theGlue);

Sets the glue for a view, giving it one of the possible types of glue.

virtual void DoSetGlue(GLUETYPE theGlue);

The “wide interface” counterpart of SetGlue. It sets the glue type of the view and then recursively sets the glue of the view’s subviews to that type.

virtual void Glue(void);

Sends a message to the view’s glue object, which in turn resizes the view according to the glue settings. XVT-Power++ calls this method automatically whenever a view’s coordinates may need updating because its enclosure has been resized.

Dragging and Stretching Methods

“Dragging” and “stretching” take effect when the user clicks on a view to select it, and then drags it or stretches it with the mouse. The last two methods in this section, SetWireFrame and GetWireFrame, pertain to CWireFrame, a helper class used by views. Wire frames are the rubberband frames that appear when you select a view for dragging or stretching. Usually, when you set a view to be draggable or sizable, the view creates a wire frame that it uses for dragging and sizing purposes. You may want to derive a modified wire frame, which draws a little differently or has a different look and feel, from the CWireFrame class.

Views can generate these events: WFSelectCmd, WFDeselectCmd, and WFMoveCmd, depending on whether a view is selected, deselected, or moved/sized. DoCommands can pass along data in the form of a void pointer so that view events can send along a pointer to the view object that is selected, deselected, or moved. Thus, you can trap the selection, deselection, or moving of views as these events happen by putting a case in the DoCommands.

virtual BOOLEAN IsDraggable(void) const;

Returns TRUE if the view can be dragged.

virtual void SetDragging(BOOLEAN isDraggable);

Sets the dragging of a view to TRUE so that the view can be dragged or to FALSE so that the view cannot be dragged.

```
virtual void DoSetDragging(BOOLEAN  
isDraggable);
```

The “wide interface” counterpart of SetDragging. It sets the dragging for the view and then recursively sets it for the view’s subviews.

```
virtual BOOLEAN IsSizable(void) const;
```

Returns a Boolean value of TRUE if the view can be sized.

```
virtual void SetSizing(BOOLEAN isSizable);
```

Takes a Boolean value that sets the sizing of a view to TRUE so that the view can be sized or to FALSE so that the view cannot be sized.

```
virtual void DoSetSizing(BOOLEAN isSizable);
```

The “wide interface” counterpart of SetSizing. It sets the sizing for the view and then recursively sets it for the view’s subviews.

```
virtual void SetWireFrame(CWireFrame*  
theNewWireFrame);
```

Sets a new wire frame for a view to use instead of the default one.

```
virtual CWireFrame* GetWireFrame(void) const;
```

Returns the wire frame of the view.

Utility Methods

```
virtual void SetId(int theID);
```

Sets the ID number of the view.

```
virtual int GetId(void) const;
```

Returns the ID number of the view.

```
virtual CWindow* GetCWindow(void) const;
```

Returns the CWindow of the view. Every view must belong to some window, and this method returns a pointer to the window enclosure of the view.

```
virtual void SetCommand(long theCommand);
```

By default, every view generates a command when it receives a mouse click. By default, some views override this command. SetCommand allows you to set the command for the view.

```
virtual long GetCommand(void) const;
```

Returns the command generated by a view when it receives a mouse click.

virtual void SetDoubleCommand(long theCommand);

By default, every view generates a double command when it receives a double mouse click. SetDoubleCommand allows you to set the double command for the view.

virtual long GetDoubleCommand(void) const;

Returns the command generated by a view when it receives a double mouse click.

virtual void SetTitle(const CString& theNewTitle);

Gives the view the title designated by theNewTitle.

virtual const CString GetTitle(void) const;

Returns the view's title. If the view does not have a title, this method returns NULL.

virtual CSubview* GetEnclosure(void);

Returns the view's enclosure, which is a CSubview object because only CSubview objects can act as enclosures for other views.

virtual void SetEnclosure(CSubview* theEnclosure);

Sets the enclosure for the view to a different enclosure. Note that if a view belongs to a certain enclosure and then it is placed inside of a different enclosure, its location relative to the window is also going to change. theEnclosure is a CSubview object because only CSubview objects can act as enclosures for other views

Printing Methods

virtual BOOLEAN DoPrint(const CRect& theRegion) const;

Notifies the view to print itself. The view sends itself to the printer on a separate page. theRegion specifies the part of the view that is to be drawn in coordinates relative to the enclosure. The part of the view that clips to this region is the part that is actually printed.

virtual void DoPrintDraw(const CRect& theRegion);

A method that is equivalent to DoDraw. It goes through the view's list of subviews notifying them to print. At the CView level, this is a "wide interface" method. At the CSubview level, there is a DoPrintDraw method that does actually go through the list of subviews and notifies them to print.

```
virtual void PrintDraw(const CRect& theRegion);
```

A method that is in charge of doing any drawing that should be sent to the printer. This drawing is just like the drawing that is done on the screen using the XVT Portability Toolkit `win_draw` functions. If you were to write your own printer `Draw` method, it would look just as if you were printing from the screen. You would still print using the view's XVT Portability Toolkit window, which can be obtained through the `CWindow` `GetXVTWindow` method. `GetXVTWindow` can return either a regular screen window or a printed window, depending on whether printing is being done. `PrintDraw` just calls the regular `Draw` method, and in most cases this is adequate. For some views, however, you may want to modify the way that drawing is done for the printer and not for the screen. In this case, you can override `PrintDraw`, putting in your own drawing specifications.

CBoss Messages and Events

```
virtual const CList* GetSubObjects(void) const;
```

A pure virtual method that must be defined by any class derived from `CBoss`, specifically, `CApplication`, `CDocument`, `CView`, and `CSubview`. It returns a list of any subobjects associated with a given object. For example, the application would return a list of all its documents, a document would return a list of all its windows, a window would return a list of all its enclosed views, a subview would return a list of all its enclosed views, and so on. At the `CView` level, this is a "wide interface" method.

```
void DoCommand(long theCommand, void* theData=NULL);
```

A method that is part of the `DoCommand` chain of events, which propagates commands up the XVT-Power++ application framework event hierarchy. The void pointer `theData` can be used to pass any user-defined structure to the `DoCommand` method.

Wide Interface Methods

The following methods have been provided for a "wide interface" so that `CView` objects can be handled just like `CSubview` objects.

```
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
```

In a nesting of views, finds the deepest subview that contains the coordinate `theLocation`. Since `CView` contains no nested subviews, this method returns `NULL`.

```
virtual CView* FindSubview(const CPoint&  
theLocation) const;
```

Finds the subview that contains the coordinate `theLocation`. If it does not find one—which it does not in the case of `CView`—it returns `NULL`.

```
virtual CView* FindSubview(int theViewId  
const);
```

This is the “wide interface” counterpart of the `FindSubview` method of `CSubview`. `CViews` do not have subviews. Thus, this method always returns `NULL`.

```
virtual CView* GetSelectedView(void) const;
```

When a view is selected, the window containing this view returns a pointer to it. The view can thus be moved and sized by the user. Since `CView` contains no nested subviews, this method returns `NULL`.

```
virtual CView* FindEventTarget(const CPoint&  
theLocation) const;
```

A method that is called to find the target of a mouse event within a subview. This virtual method is defined to return the deepest subview at the global coordinates of `theLocation`. This method can be overridden by a subview in order to trap all events regardless of the subviews it contains or to change the event targeting algorithm. For example, a list box contains many subviews (text items). When you click on a text item inside a list box, the event does not go to the `CText` view. Instead, the list box overrides `FindEventTarget` and traps the event by returning this (itself) as the target for the mouse events received. Then, it treats the mouse events as necessary, without sending them on to its subviews.

```
virtual const CList* GetSubviews(void) const;
```

This is the “wide interface” counterpart of the `GetSubviews` method of `CSubview`. `CViews` do not have subviews. Thus, this method always returns `NULL`.

Event Handler Methods

```
virtual void DoTimer(long theTimerId);
```

A method that is called when the XVT Portability Toolkit generates a timer (`E_TIMER`) event. You set a timer using the XVT Portability Toolkit’s `set_timer` function, which takes a window and a time interval (in milliseconds) and returns an ID number for the timer. This is the ID number that is passed to the `theTimerId` of `DoTimer`. When the timer generates a timer

event, CSwitchBoard passes it to the XVT Portability Toolkit-designated window via DoTimer. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

```
virtual void DoUser(long theUserId, void* theData);
```

A method that is called when the XVT Portability Toolkit generates a user (E_USER) event. As noted in the *XVT Programmer's Guide*, user events allow you to pass custom events to windows and dialogs. theUserId is the ID number that designates a particular kind of user event. The void pointer theData can be used to pass any user-defined structure to DoUser, just as you would for a DoCommand. As with timer events, CSwitchBoard passes a user event to a designated window. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

Protected Methods

```
BOOLEAN IView(BOOLEAN isVisible = TRUE,  
GLUETYPE theGlue = NULLSTICKY);
```

The initializer, which takes a visibility state and a glue type.

Utility

```
void Prepare(const CRect& theClippingRegion);
```

Prepares the view for drawing, basically by setting its clipping region.

Private Methods

```
void BuildView(CSubview* theEnclosure, const  
CRect& theRegion);
```

A method that is called during construction and equal operations. It takes care of building the view.

```
void CopyView(const CView& theView);
```

A method that is called during construction and equal operations. It takes care of copying the view.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
virtual void UpdateUnits(CUnits* theUnits);
```

Summary: CView.h

```
#ifndef CView_H
#define CView_H

#include "PwrDef.h"
#include CEnvironment_i
#include CBoss_i
#include CPoint_i
#include CRect_i
#include CList_i
#include CString_i

class CNativeView;
class CSubview;
class CWindow;
class CGlue;
class CWireFrame;

class CView : public CBoss
{
public:
    // construct, destruct
    CView(CSubview* theEnclosure, const CRect& theRegion);
    CView(const CView& theView);
    CView& operator=(const CView& theView);
    virtual ~CView(void);

    BOOLEAN IView(BOOLEAN isVisible = TRUE,
        GLUETYPE theGlue = NULLSTICKY);

    // visibility state:
    virtual BOOLEAN IsVisible (void) const;

    virtual void Show(void);
    virtual void Hide(void);

    virtual void DoShow(void);
    virtual void DoHide(void);

    // activate events (i.e., is this the active view or not)
    virtual BOOLEAN IsActive(void) const;

    virtual void Deactivate(void);
    virtual void Activate(void);

    virtual void DoActivate(void);
    virtual void DoDeactivate(void);

    // enable/disable events (i.e., receive events or not)
    virtual BOOLEAN IsEnabled(void) const;
```

```

virtual void Disable(void);
virtual void Enable(void);

virtual void DoEnable(void);
virtual void DoDisable(void);

// mouse events:
virtual void MouseDouble(CPoint theLocation,short theButton = 0,
                        BOOLEAN isShiftKey=FALSE,
                        BOOLEAN isControlKey=FALSE);
virtual voidMouseDown(CPoint theLocation,short theButton = 0,
                     BOOLEAN isShiftKey=FALSE,
                     BOOLEAN isControlKey=FALSE);
virtual voidMouseMove(CPoint theLocation,short theButton = 0,
                     BOOLEAN isShiftKey=FALSE,
                     BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0,
                    BOOLEAN isShiftKey=FALSE,
                    BOOLEAN isControlKey=FALSE);

virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,
                          BOOLEAN isShiftKey=FALSE,
                          BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,
                        BOOLEAN isShiftKey=FALSE,
                        BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,
                        BOOLEAN isShiftKey=FALSE,
                        BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,
                      BOOLEAN isShiftKey=FALSE,
                      BOOLEAN isControlKey=FALSE);

virtual CView* FindHitView(const CPoint& theLocation) const;

// Drawing:
virtual void Draw(const CRect& theClippingRegion);
virtual void DoDraw(const CRect& theClippingRegion);

virtual void Draw(void);
virtual void DoDraw(void);

// Other events:
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);

virtual void Size(const CRect& theNewSize);
virtual void DoSize(const CRect& theNewSize);

virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont,
                      BOOLEAN isUpdate = FALSE);

virtual void HScroll(SCROLL_CONTROL theEventType,
                    UNITS thePosition);
virtual void VScroll(SCROLL_CONTROL theEventType,
                    UNITS thePosition);

// Coordinates and locations:
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CRect GetClippedFrame(void) const;
virtual CRect GetLocalFrame(void) const;

```



```

virtual CPoint GetOrigin(void) const;
virtual CPoint GetGlobalOrigin(void) const;

virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);

// Environment:
virtual const CEnvironment* GetEnvironment(void) const;

virtual void SetEnvironment(const CEnvironment& theNewEnvironment,
    BOOLEAN isUpdate = FALSE);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment,
    BOOLEAN isUpdate = FALSE);

virtual BOOLEAN IsEnvironmentShared(void);

// View utility:
virtual void SetId(int theID);
virtual int GetId(void) const;

// Glue functions
virtual GLUETYPE GetGlue(void) const;
virtual void SetGlue(GLUETYPE theGlue);
virtual void DoSetGlue(GLUETYPE theGlue);

virtual void Glue();

// Dragging and stretching functions:
virtual BOOLEAN IsDraggable(void) const;
virtual void SetDragging(BOOLEAN isDraggable);
virtual void DoSetDragging(BOOLEAN isDraggable);

virtual BOOLEAN IsSizable(void) const;
virtual void SetSizing(BOOLEAN isSizable);
virtual void DoSetSizing(BOOLEAN isSizable);

virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual CWireFrame* GetWireFrame(void) const;

// Utility:
virtual CWindow* GetCWindow(void) const;

virtual void SetCommand(long theCommand);
virtual long GetCommand(void) const;

virtual void SetDoubleCommand(long theCommand);
virtual long GetDoubleCommand(void) const;

virtual void SetTitle(const CString& theNewTitle);
virtual const CString GetTitle(void) const;

virtual CSubview* GetEnclosure(void);
virtual void SetEnclosure(CSubview* theEnclosure);

// Printing Methods:
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void DoPrintDraw(const CRect& theRegion);
virtual void PrintDraw(const CRect& theRegion);

// CBoss Messages and events:
virtual const CList* GetSubObjects(void) const;
virtual void DoCommand(long theCommand, void* theData = NULL);

```

```

// Fat interface:
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* GetSelectedView(void) const;
virtual CView* FindEventTarget(const CPoint& theLocation) const;
virtual const CList* GetSubviews(void) const;

//Other event handler methods
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);

protected:
    CRect itsFrame; // its rectangular border
    CPoint itsOrigin; // its origin for scrolling
    CWindow* itsCWindow; // the window in which it belongs
    long itsCommand; // command to generate on mouse-click
    long itsDoubleCommand; // command generated on a mouse-double-click
    CSubview* itsEnclosure; // the enclosure to which the object belongs

    // Utility:
    void Prepare(const CRect& theClippingRegion);

private:
    CEnvironment* itsEnvironment; // its local environment
    BOOLEAN itIsVisible; // whether it is visible within the frame
    BOOLEAN itIsEnabled; // can the view receive events
    BOOLEAN itIsActive; // is this view in an active state
    CString itsTitle; // the View's title
    int itsId; // the View's id
    CGlue* itsGlue; // the glue
    CWireFrame* itsWireFrame; // wire frame used for sizing and dragging

    void BuildView(CSubview* theEnclosure, const CRect& theRegion);
    void CopyView(const CView& theView);
};

#endif CView_H

```

CVirtualFrame

CSubview — **CVirtualFrame** — CScroller — CListBox

Description

CVirtualFrame is an abstract class that represents an area on the screen with a virtual size that is larger than its display area. Every virtual frame thus has two regions associated with it: a virtual region and a real visible region (display area) that is located inside a window or some other view. A CVirtualFrame object is a subview, so it can contain other nested views.

Heritage

Superclass: CSubview

Subclass: CScroller

Usage

Objects are placed inside of a CVirtualFrame and registered as subviews. These objects are displayed inside the viewing rectangle relative to a virtual origin. As the origin changes, different areas of the virtual frame are displayed, revealing the subviews contained in the virtual frame.

You can set the virtual size of the frame manually using SetVirtualFrame. If this is not done, the virtual frame automatically sizes itself to enclose all of its subviews.

This is an abstract class that must be extended by derived classes that manipulate the scrolling by updating the origin (through the use of NScrollBars, for example).

Public Data Members

None.

Protected Data Members

None.

Private Data Members

CPoint	*itsViewOrigin;	the origin of the virtual frame relative to the viewable rectangle
CRect	*itsVirtualFrame;	the virtual region
static double	itsXProportion;	internal sizing variable
static double	itsYProportion;	internal sizing variable

Public Methods

Initializer and Destructor Methods

```
BOOLEAN IVirtualFrame(BOOLEAN isVisible = TRUE,
    GLUETYPE theGlue = NULLSTICKY);
```

The initializer, which takes a visibility state and a glue type.

```
~CVirtualFrame(void);
```

The destructor. It cleans up after the virtual frame and deletes any views nested within it.

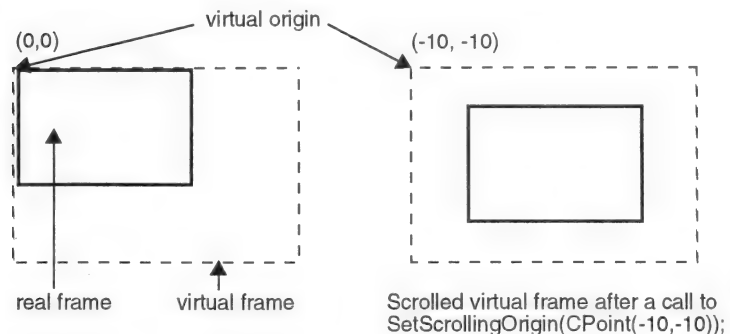
Methods Pertaining to the Scrolling Origin

```
void SetScrollingOrigin(const CPoint&
    theNewOrigin);
```

Sets the actual position of the scrolling area of the virtual frame, relative to the entire frame.

```
void const CPoint& GetScrollingOrigin(void);
```

Returns a coordinate (CPoint) indicating the actual position of the scrolling area of the virtual frame, relative to the entire frame.



Sizing Methods

```
virtual void SetVirtualFrame(UNITS theNewWidth,  
                             UNITS theNewHeight);
```

Sets the width and height of the virtual frame's virtual region.

```
virtual CRect GetVirtualFrame(void) const;
```

Returns a rectangle (CRect) describing the dimensions of the virtual frame's virtual region. The rectangle is top-left justified; that is, its top-left point is 0,0, and the bottom right point represents the height and the width.

```
virtual void EnlargeToFit(const CRect&  
                           theRegionToInclude);
```

Takes a region that is to be included in the virtual frame and enlarges the virtual area of the frame so that it encompasses this region. This region has coordinates relative to the *visible area* of the virtual frame.

```
virtual void ShrinkToFit(void);
```

Shrinks the virtual region of the frame so that it is just large enough to fit all of the subviews contained in the virtual frame.

Inherited Utilities

```
virtual void Draw(const CRect&  
                  theClippingRegion);
```

An overridden method that takes care of any drawing that must be done by the visible region of the virtual frame.

theClippingRegion is the portion of the visible region that must be drawn, and it is represented in global, window-relative coordinates.

```
virtual void Size(const CRect& theNewSize);
```

An overridden method that takes care of some special sizing implementations that are specific to the virtual frame. theNewSize specifies the size of the virtual region of the frame. The reset region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, theNewSize, are relative to the enclosure—like the coordinates of the region that is passed in when a view is instantiated.

Protected Methods

Constructor Methods

```
CVirtualFrame(CSubview* theEnclosure, const  
CRect& theRegion, UNITS theVirtualWidth=0,  
UNITS theVirtualHeight=0);
```

A constructor. *theEnclosure* is a pointer to the subview that will contain the virtual frame. *theRegion* is the actual visible region of the virtual frame. It is a coordinate location, local to *theEnclosure*, that is used to place the visible region of the virtual frame. The parameters *theVirtualWidth* and *theVirtualHeight* together define the size of the *virtual* area.

```
CVirtualFrame(const CVirtualFrame&  
theVirtualFrame);
```

A copy constructor that duplicates the attributes, but not the contents, of a virtual frame. That is, it copies the virtual frame but does not do a deep copy of any views nested within it.

```
CVirtualFrame& operator=(const CVirtualFrame&  
theVirtualFrame);
```

An assignment operator that duplicates the attributes, but not the contents, of a virtual frame. That is, it copies the virtual frame but does not do a deep copy of any views nested within it.

Scrolling Method

```
virtual void ScrollViews(const CPoint&  
theNewOrigin);
```

Scrolls the views contained in the virtual frame, given a new virtual origin.

Inherited Methods

```
virtual void AddSubview(const CView*  
theSubview);
```

An overridden method that adds a pointer to a *CSubview*-derived object to the contents of the virtual frame. This method ensures that the virtual region of the frame is large enough to include the newly added object.

Private Methods

```
virtual void PreSize(void);
```

A hook for future implementation.

```
virtual void PostSize(void);
```

A hook for future implementation.

Scroll Range

The following two methods set the minimal, maximal, and current virtual position. Derived classes must provide these methods and respond appropriately. For example, the scroller updates its scrollbars.

```
virtual void SetHScrollRange(UNITS theTop,  
    UNITS theBottom, UNITS thePosition)=0;
```

Depending on the position of the virtual frame, sets the range of horizontal scrolling

```
virtual void SetVScrollRange(UNITS theTop, UNITS  
    theBottom, UNITS thePosition)=0;
```

Depending on the position of the virtual frame, sets the range of vertical scrolling.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);  
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
    isControlKey);  
virtual CUnits* GetUnits(void) const;  
virtual void SetUnits (CUnits* theCoordinateUnits);  
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);  
virtual void Deactivate(void);  
virtual void Disable(void);  
virtual void DoCommand(long theCommand, void* theData=NULL);  
virtual BOOLEAN DoPrint(const CRect& theRegion) const;  
virtual void DoTimer(long theTimerId);  
virtual void DoUser(long theUserId, void* theData);  
virtual void Draw(void);  
virtual void Enable(void);  
virtual CView* FindHitView(const CPoint& theLocation) const;  
virtual CRect GetClippedFrame(void) const;  
virtual long GetCommand(void) const;
```

```
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual voidMouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
```



```
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CSubview

```
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
```

```

virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjscts(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);
virtual void SetSelectedView(CView* theSelectedView);

```

Summary: CVirtualFrame.h

```

#ifndef CVirtualFrame_H
#define CVirtualFrame_H

#include "CSubview.h"

class CVirtualFrame :public CSubview
{
public:
    // Constructors, destructors, and initializers
    BOOLEAN IVirtualFrame(BOOLEAN isVisible = TRUE,
        GLUTYPE theGlue = NULLSTICKY);
    virtual ~CVirtualFrame(void);

    // Scrolling functions:
    virtual void ScrollViews(const CPoint& theNewOrigin);

    // Sizing functions:
    virtual void SetVirtualFrame(UNITS theNewWidth,
        UNITS theNewHeight);
    virtual CRect GetVirtualFrame(void) const;

    virtual void EnlargeToFit(const CRect& theRegionToInclude);
    virtual void ShrinkToFit(void);

    // Inherited utilities:
    virtual void Draw(const CRect& theClippingRegion);
    virtual void Size(const CRect& theNewSize);

protected:

```

```
// Constructors:
CVirtualFrame(CSubview* theEnclosure, const CRect& theRegion,
              UNITS theVirtualWidth=0, UNITS theVirtualHeight=0);
CVirtualFrame(const CVirtualFrame& theVirtualFrame);
CVirtualFrame& operator=(const CVirtualFrame& theVirtualFrame);

// Inherited:
virtual void AddSubview(const CView* theSubview);

void SetScrollingOrigin(const CPoint& theNewOrigin);
CPoint GetScrollingOrigin(void);

private:

    friend class CScroller;

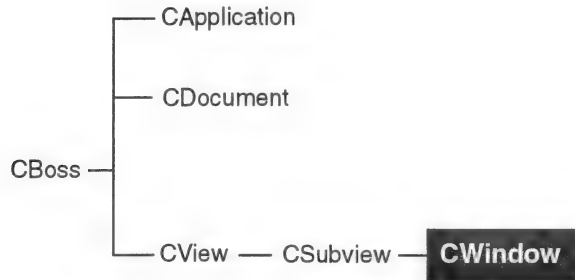
    CPoint *itsViewOrigin;           // the origin of the virtual frame
    CRect *itsVirtualFrame;         // the virtual region

    static double itsXProportion;    // internal sizing variable
    static double itsYProportion;    // internal sizing variable

    virtual void SetHScrollRange(UNITS theLeft,UNITS theRight,
                                UNITS thePosition)=0;
    virtual void SetVScrollRange(UNITS theTop,UNITS theBottom,
                                UNITS thePosition)=0;
};

#endif CVirtualFrame_H
```

CWindow



Description

CWindow objects consist of a physical window displayed on the screen. CWindow is a special type of CSubview and thus may contain other CView objects. The CWindow object is the topmost view in the nesting of views; it is the link between the views and the documents. A CWindow object can be of any XVT Portability Toolkit type and receives all window events. Most of the window management, such as moving and sizing, is done by the window manager or the XVT-Power++ desktop. Possible XVT Portability Toolkit window types are shown in the following table. For further information, see the *XVT Programmer's Guide*.

W_DOC	document window
W_PLAIN	single-bordered window
W_DBL	double-bordered window
W_NONE	no std_win ever
W_NO_BORDER	no border

Heritage

Superclass: CSubview

Subclass: CShellWin

Usage

This is a class from which application-specific derived classes are built. To use it, it is sufficient to derive a new class, create it, and add subviews or native views. For an example of a class that has been

derived from CWindow, refer to CShellWin. See CDesktop for information on managing windows and their layout on the screen.

Public Data Members

None.

Protected Data Members

CDocument	*itsDocument;	the window's boss
NWinScrollBar	*itsVScrollBar;	vertical scrollbar
NWinScrollBar	*itsHScrollBar;	horizontal scrollbar

Private Data Members

BOOLEAN	itDrawsBackground;	whether the window has a background
BOOLEAN	itIsClosable;	whether the window can be closed
BOOLEAN	itIsSizable;	whether the window can be sized
WINDOW	itsXVTWindow;	the window's XVT Portability Toolkit window handle
int	itsMenuBarId;	the window's URL menubar ID
long	itsAttributes;	the window's XVT Portability Toolkit attributes

Friends

friend class	CSwitchBoard;	allows access to the window event methods
friend class	CTaskWin;	allows access to itsXVTWindow
friend class	CView;	allows access for speed

Methods

Constructor and Initializer Methods

```
CWindow(CDocument* theDocument, const CRect&
        theRegion, const CString&= NULLString,
        long theWindowAttributes = WSF_NONE,
```

```
WIN_TYPE theWindowType    = W_DOC,
int theMenuBarId          = MENU_BAR_RID);
```

A constructor. `theDocument` is a pointer to the window's document. The coordinates of `theRegion` are screen-relative coordinates, and their meaning depends upon the platform. Some platforms honor these coordinates, and others do not; however, the size of the `CRect` is always important. `theWindowAttributes` takes a value from a set of XVT Portability Toolkit-provided attributes that you can give to windows. You can OR together the appropriate flags into an attribute value. The possible flags for `theWindowAttributes` are shown in the following table. The XVT Portability Toolkit supplies other, platform-specific flags.

WSF_NONE	no flags set
WSF_SIZE	is user-sizeable
WSF_CLOSE	is user_closeable
WSF_HSCROLL	has horizontal scrollbar outside client area
WSF_VSCROLL	has vertical scrollbar outside client area
WSF_DECORATED	all of above four flags are set
WSF_INVISIBLE	is initially invisible
WSF_DISABLED	is initially disabled
WSF_ICONIZABLE	is iconizable
WSF_ICONIZED	is initially iconized
WSF_NO_MENUBAR	has no menubar of its own
WSF_MAXIMIZED	initially maximized

`theWindowType` specifies the type of the window. For a listing of the possible types, refer to the table at the beginning of this discussion of `CWindow`. Finally, `theMenuBarId` is the resource ID number of the menubar, which must be a valid URL-based resource.

```
CWindow(CDocument* theDocument, WINDOW
theXVTWindow);
```

```
CWindow(const CWindow& theWindow);
```

A copy constructor that duplicates the attributes of the window but does not copy its subviews, views, and native views.

```

BOOLEAN IWindow(BOOLEAN isBackgroundDrawn = TRUE,  

                  const CString theTitle      = NULLString,  

                  BOOLEAN isVisible          = TRUE);

```

The initializer. The Boolean flag `isBackgroundDrawn` determines whether the window draws a background using its environment. For example, windows are normally all the same color, and you can set the color in ways provided by your particular platform—such as Motif, Open Look, Macintosh, and so on. If you want to create windows of different colors, you must create a window with the `isBackgroundDrawn` parameter set to `TRUE` and then set the window's environment to the color of your choice. If do not want the extra drawing because you just want to use the basic window color, set this parameter to `FALSE`. The next two parameters are the title of the window and a flag indicating whether the window is visible.

Methods for Message Passing

```

virtual void DoCommand(long theCommand, void*  

                  theData=NULL);

```

A method that is overridden for any windows that need to handle `DoCommands`. It ensures that if the window itself does not handle the `DoCommand`, the event is sent up to the window's document. Even if an object inherits from this class and overrides the `DoCommand`, it uses the window's `DoCommand` by default. If you do not want to process a certain command number, call the window's `DoCommand`. The void pointer `theData` can be used to pass any user-defined structure to the `DoCommand` method.

```

virtual void UpdateMenus(void);

```

Updates the menubar so that it reflects the correct settings for the window. You must override this method, specifying how the window uses the menubar. This method is called internally at different places and can be called by the user as well. When the window is brought to the front, for example, `UpdateMenus` is called automatically.

Window Setup and Closing

```

virtual BOOLEAN Close(void);

```

Closes the window and also has several side effects. This method is not only called directly but is often called when a user physically closes a window. `Close` first checks to see whether the document that is about to be closed needs saving. If it does, then `Close` invokes a dialog box that gives the user the option of saving the data or cancelling. If the user selects "Yes", then

Close calls the DoSave method of the window's document. If the user selects "No", then DoSave is not called and the window is closed. If the user elects to cancel, then the closing mechanism is cancelled and the program returns to normal.

The closing of the window generates an XVT Portability Toolkit event that has the effect of deleting whatever CWindow Close is called upon. Calling Close on a window is synonymous with deleting it. If any pointers are pointing to the window object, those pointers are now pointing to deleted memory. Thus it is important that nothing else be done with the CWindow object after a window is closed. Once it is closed, it is gone and all memory of it is deleted.

virtual WINDOW GetXVTWindow(void) const;

Returns an XVT Portability Toolkit type of window handle. This method is useful if you want to manipulate the window at the XVT Portability Toolkit level.

virtual BOOLEAN IsClosable(void) const;

Returns a Boolean value of TRUE if the window can be closed.

virtual BOOLEAN IsSizable(void) const;

Returns a Boolean value of TRUE if the window can be sized.

virtual CDocument* GetDocument(void);

Returns a pointer to the window's document.

Window Background Methods

virtual BOOLEAN IsBackgroundDrawn(void) const;

Returns a Boolean value of TRUE if the window has a background.

virtual void SetBackgroundDrawing(BOOLEAN isBackgroundDrawn);

Allows you to set a window's background by giving it a Boolean value of TRUE or FALSE.

Window Scrollbar Methods

virtual NWinScrollBar* GetVScrollBar(void) const;

Returns the window's attached vertical scrollbar.

virtual NWinScrollBar* GetHScrollBar(void) const;

Returns the window's attached horizontal scrollbar.

Inherited Utility Methods

CWindow overrides the following basic inherited event methods.

**virtual void Key(int theKey, BOOLEAN
isShiftKey, BOOLEAN isControlKey);**

A method that is called to send a keyboard event to a window. By default, this event is passed up to the window's CDocument. `int` is the ASCII value for the character key that was pressed; `isShiftKey` and `isControlKey` indicate whether the SHIFT or CONTROL key was pressed in conjunction with the character key.

virtual void Size(const CRect& theNewSize);

Sizes the window according to the coordinates of `theNewSize`.

**virtual void Draw(const CRect&
theClippingRegion);**

Draws the background for the window if the `isBackgroundDrawn` flag has been set to TRUE. `theClippingRegion` is the part of the window that needs to be refreshed; if it is set to NULL, the entire window is drawn. `theClippingRegion` is in global, window-relative coordinates. This method is automatically called after an E_UPDATE event.

virtual void Hide(void);

Makes the window invisible.

virtual void Show(void);

Makes the window visible.

virtual void Enable(void);

Enables the window so that it can receive events.

virtual void Disable(void);

Disables the window so that it cannot receive events.

**virtual void SetEnclosure(CSubview*
theEnclosure);**

A CSubview method that allows you to move a subview object from one enclosure to another. However, windows are special. This method has been disabled by overriding, which ensures that you cannot call `SetEnclosure` for a window because windows do not have an enclosure. On some platforms, a window might have a task window as an enclosure, but this is irrelevant to XVT-Power++.

```
virtual void SetTitle(const CString&  
theNewTitle);
```

Sets the window's title, which is a CString.

```
virtual void SetSelectedView(CView  
*theSubview);
```

Sets a window's selected view. Any subview can have a selected view set, and SetSelectedView has been overridden for CWindow. The default behavior for windows (which can be overridden) takes the selected view's font and updates the Font menu to reflect that font. Font menus usually have a certain font type and size checked or otherwise marked to indicate that this font is active.

```
virtual CRect GetClippedFrame(void) const;
```

Views are always clipped to their enclosures, so portions of a view may not be visible. This method returns a region (CRect) with the coordinates of the visible portion of a clipped view. These coordinates are global, that is, window-relative. Usually, you will call GetClippedFrame before you call DoDraw in order to pass the smallest region possible to DoDraw. Because Draw and DoDraw take window-relative coordinates, GetClippedFrame returns a window-relative CRect.

```
virtual const CEnvironment*  
GetEnvironment(void) const;
```

For purposes of consistency, conciseness, and ease of use, views often share environments—with their enclosure, their window, their document, or their application. This method tests to see if a window is using its own private environment. If not, it checks the environment of the window's document. If the document is not using an environment of its own, GetEnvironment recurses upward until it finds some environment that is being used and returns it. At the top level the default environment of the CApplication object is always available.

```
virtual void DoMenuCommand(MENU_TAG  
theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);
```

Handles the event that occurs when the user selects an item from the menubar. DoMenuCommand first goes to the window from which the selection was made. If the window cannot handle the command, the event chains upward, stopping at any point, to the document and on up to the application. theMenuItem is the menu item number of the menu item

resource ID as specified in the URL menu definition. `isShiftKey` and `isControlKey` specify whether the mouse is used with the SHIFT key or CONTROL key.

This method should be overridden to handle a menubar event. If not, this event should be called so that the menu command can be propagated up to the document.

```
virtual void ChangeFont(FONT theNewFont,  
FONT_PART thePart);
```

When the user selects a font from the menubar, `ChangeFont` sends a font object, `theFont`, that indicates the current state of the Font/Style menu. The specific font property that was changed is indicated in `thePart`, which is of type `FONT_PART`, an XVT Portability Toolkit variable that is defined as follows:

```
typedef enum { /* symbol for part of font desc. */  
F_STYLE; /* style (bold, italic, ect.) */  
F_FAMILY; /* FAMILY (Times, etc.) */  
F_SIZE; /* point size part */  
} FONT_PART;
```

The window verifies whether it has a selected view, and if it does, it sends the event to the selected view. If not, it sends the event to itself—only if this window actually has an environment of its own; if the window is sharing an environment with its document, `ChangeFont` propagates the event up to the document level. This event goes directly to the application if no windows are up.

Event Methods (called by event handlers only)

```
virtual void SizeWindow(int theWidth, int  
theHeight);
```

A method that is called internally when the window is sized by the user.

```
virtual void DoControl(int  
theControlID, CONTROL_INFO theControlInfo);
```

A method that is called by the switchboard when it receives a native view event. The window checks its list of native views for the ID that matches `theControlID`. If it finds the native view on its list, it passes a `DoHit` message directly to it. If it does not find a matching native view, it sends a message to its subviews to search for it. Usually, this method is called automatically, although you can call it if desired.

`CONTROL_INFO` is an XVT Portability Toolkit structure for complicated native views such as scrollbars, which need more information than a unique ID number in order to handle a `DoHit`

event—information about the window type, the site of the activity, the thumb position, and so on.

virtual void DoActivateWindow(void);

A method that is called internally whenever a window is called to the front of a stack of windows. It can be overridden to do whatever you want it to do when a window is made active and can receive events.

virtual void DoDeactivateWindow(void);

A method that is called internally whenever a window is called to the front of a stack of windows. It can be overridden to do whatever you want it to do when a window is made inactive and cannot receive events.

virtual void DoTimer(long theTimerId);

A method that is called when the XVT Portability Toolkit generates a timer (E_TIMER) event. You set a timer using the XVT Portability Toolkit's `set_timer` function, which takes a window and a time interval (in milliseconds) and returns an ID number for the timer. This is the ID number that is passed to the `theTimerId` of `DoTimer`. When the timer generates a timer event, `CSwitchBoard` passes it to the XVT Portability Toolkit-designated window via `DoTimer`. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

virtual void DoUser(long theUserId, void* theData);

A method that is called when the XVT Portability Toolkit generates a user (E_USER) event. As noted in the *XVT Programmer's Guide*, user events allow you to pass custom events to windows and dialogs. `theUserId` is the ID number that designates a particular kind of user event. The void pointer `theData` can be used to pass any user-defined structure to `DoUser`, just as you would for a `DoCommand`. As with timer events, `CSwitchBoard` passes a user event to a designated window. The window passes it on to its selected view, if it has one. The timer event may propagate upwards from a view to the window to the document and all the way to the application object, stopping at any point.

Protected Methods

CWindow& operator=(const CWindow& theWindow);

An assignment operator that duplicates the attributes of the window but does not do a deep copy of its subviews, views, and native views.

virtual ~CWindow (void);

The destructor. It deletes (closes) the window and removes the representation of it from the screen.

Private Methods

Scrolling Events

void DoHScroll(SCROLL_CONTROL theEvent, short thePos);

A method that is called internally when a window with scrollbars is scrolled horizontally. SCROLL_CONTROL is the site of the activity, and thePos is the position of the scrollbar's thumb.

void DoVScroll(SCROLL_CONTROL theEvent, short thePos);

A method that is called internally when a window with scrollbars is scrolled vertically. SCROLL_CONTROL is the site of the activity, and thePos is the position of the scrollbar's thumb.

virtual void UpdateUnits(CUnits* theUnits);

Updates the CUnits object indicated by theUnits, which is the object owned by the CWindow object. The owner updates itself and propagates the update message to any of its child objects. That is, the window updates itself and propagates the update message to all of its enclosed views, but only if they are sharing the same CUnits object. Basically, the owner of a CUnits object is the topmost object in the hierarchy that is using the CUnits object.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

virtual CUnits* GetUnits(void) const;

virtual void SetUnits (CUnits* theCoordinateUnits);

From CView

virtual void Activate(void);

```
virtual void Deactivate(void);
virtual BOOLEAN DoPrint(const CRect& theRegion) const;
virtual void Draw(void);
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void PrintDraw(const CRect& theRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
```

```
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CSubview

```
virtual void AddSubview(const CView* theSubview);
virtual CPoint AutoScroll(int theHorizontalChange, int theVerticalChange);
virtual void DoActivate(void);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoPrintDraw(const CRect& theClippingRegion);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
```

```

virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual FindEventTarget(const CPoint& theLocation) const;
virtual CView* FindSubview(int theViewId) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual void FindSubviews(const CPoint& theLocation, CList* theList) const;
CView* GetKeyFocus(void) const;
int GetNumViews(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjscts(void) const = NULL;
virtual const CList* GetSubviews(void) const;
virtual void PlaceBottomSubview(const CView* theView);
virtual void PlaceTopSubview(const CView* theView);
virtual void RemoveSubview(const CView* theSubview);
virtual void SetKeyFocus(CView *theFocusedView);

```

Overrides

IWindow, ~Window, and any other virtual function as needed.

Summary: CWindow.h

```

#ifndef CWindow_H
#define CWindow_H

#include "CSubview.h"

class CDocument;

class CWindow : public CSubview
{
public:
    // Constructors, destructors:

    // This is the old retired construtor:
    CWindow(CDocument *theDocument,
            const CRect& theRegion,
            BOOLEAN isClosable,
            BOOLEAN isSizable,
            WIN_TYPE theWindowType);

    // This is the new XVT3 constructor:
    CWindow(CDocument* theDocument,
            const CRect& theRegion,
            const CString&           = NULLString,
            long theWindowAttributes = WSF_NONE,
            WIN_TYPE theWindowType   = W_DOC,
            int theMenuBarId         = MENU_BAR_RID);

    CWindow(CDocument* theDocument, WINDOW theXVTWindow);
    CWindow(const CWindow& theWindow);

```



```

    BOOLEAN IWindow(BOOLEAN isBackgroundDrawn    = TRUE,
                    const CString theTitle        = NULLString,
                    BOOLEAN isVisible             = TRUE);

// Messages:

virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void UpdateMenus(void);

// Window setup and closing:
virtual BOOLEAN Close(void);

virtual WINDOW GetXVTWindow(void) const;

virtual BOOLEAN IsClosable(void) const;
virtual BOOLEAN IsSizable(void) const;

virtual CDocument* GetDocument(void);

// Window background:
virtual BOOLEAN IsBackgroundDrawn(void) const;
virtual void SetBackgroundDrawing(BOOLEAN isBackgroundDrawn);

// Window's ScrollBars:
virtual NWinScrollBar* GetVScrollBar(void) const;
virtual NWinScrollBar* GetHScrollBar(void) const;

// Inherited utility:
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void Size(const CRect& theNewSize);
virtual void Draw(const CRect& theClippingRegion);
virtual void Hide(void);
virtual void Show(void);

virtual void Enable(void);
virtual void Disable(void);

virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetSelectedView(CView *theSubview);

virtual CRect GetClippedFrame(void) const;
virtual const CEnvironment* GetEnvironment(void) const;

virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey,
                           BOOLEAN isControlKey);
virtual void ChangeFont(FONT theNewFont, FONT_PART thePart);

// Events (called by event handlers only):

virtual void SizeWindow(int theWidth, int theHeight);
virtual void DoControl(int theControlID, CONTROL_INFO theControlInfo);
virtual void DoActivateWindow(void);
virtual void DoDeactivateWindow(void);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);

protected:
    CDocument *itsDocument; // its boss
    NWinScrollBar* itsVScrollBar; // the window's vertical scrollbar
    NWinScrollBar* itsHScrollBar; // the window's horizontal scrollbar

    CWindow& operator=(const CWindow& theWindow);
    virtual ~CWindow (void);

```

```
virtual void DoHScroll(SCROLL_CONTROL theEvent, short thePos);
virtual void DoVScroll(SCROLL_CONTROL theEvent, short thePos);

virtual void UpdateUnits(CUnits* theUnits);

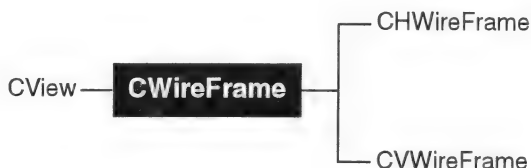
private:
    friend class CSwitchBoard; // Allow access to Window event methods
    friend class CTaskWin;    // Allow access to itsXVTWindow
    friend class CView;       // Allow access for speed

    BOOLEAN itDrawsBackground; // if the window wants a background
    BOOLEAN itIsClosable;      // window may be closed
    BOOLEAN itIsSizable;       // window may be sized
    WINDOW itsXVTWindow;       // its XVT Portability Toolkit WINDOW
    int itsMenuBarId;          // its URL menubar ID
    long itsAttributes;         // its XVT Portability Toolkit attributes

    static EVENT_HANDLER WINDOWHandler; // The global default event
                                        // handler.
};

#endif CWindow_H
```

CWireFrame



Description

CWireFrame provides a type of CView that acts as a helper class to all other classes in the CView hierarchy. It is a rectangular rubberband frame that surrounds a CView object, enabling it to be dragged and sized with the mouse.

Heritage

Superclass: CView

Usage

A wire frame is normally not instantiated directly; instead, it is created automatically by CView when necessary. Typically, you do not need to be aware of this class since it is used internally. However, CWireFrame is documented here in case you want to override this class and modify it so that the wire frame draws differently or alter the way that the rubberbanding, sizing, and dragging are implemented. If you do this, you can set a view's wire frame by calling CView::SetWireFrame.

Views normally use a wire frame when the user enables the sizing or dragging properties. If several views have wire frames, this class automatically takes care of selecting and deselecting views as the user clicks on various views. In order to select more than one view at a time, the user must hold down the SHIFT key while selecting or click the right mouse button on the view (multi-button mouse). Finally, if multiple views are selected, they move as a group when the user drags the mouse.

Public Data Members

static const int WIRESIZE; the size difference between the wire frame and itsOwner

Protected Data Members

STATE	itsState;	the current event state
DRAWSTATE	itsDrawState;	internal drawing state flag
CPoint	itsGrabPoint;	the point that was grabbed
long	itsSelectCommand;	command generated upon selection
long	itsDeselectCommand;	command generated upon deselection
long	itsSizeCommand;	command generated upon moving/sizing

Private Data Members

long	itsSizeSide;	the side to be sized
CRect	itsMinimalRegion;	the smallest size possible
BOOLEAN	itIsDraggable;	whether the wire frame can receive dragging events
BOOLEAN	itIsSizable;	whether the wire frame can receive sizing events
PNT*	itsFramePnts;	the ending point of the sizing area
CView*	itsOwner;	the CView it sizes
BOOLEAN	itsDrawState;	internal drawing state flag
BOOLEAN	itsGrabState;	internal grabber state flag
BOOLEAN	isGroupMode;	whether the wire frame is in group mode
UNITS	itsGroupLeft;	group's left boundary
UNITS	itsGroupRight;	group's right boundary
UNITS	itsGroupTop;	group's top boundary
UNITS	itsGroupBottom;	group's bottom boundary

Public Methods

Constructor, Destructor, and Initializer Methods

```
CWireFrame(CView *theOwner, CSubview*
theGroupEnclosure = NULL);
```

A constructor. It simply takes a pointer to its owner, the CView object with which it is associated. It gets information about this object, such as the object's enclosure, from the object itself. theGroupEnclosure is the subview that acts as the enclosure of the group containing this wire frame.

```
CWireFrame(const CWireFrame& theFrame);
```

A copy constructor, which duplicates the attributes of a wire frame, including ownership.

```
CWireFrame& operator=(const CWireFrame& theFrame);
```

An assignment operator, which duplicates the attributes of a wire frame, including ownership.

```
virtual ~CWireFrame(void);
```

The destructor, which cleans up after the wire frame.

```
virtual BOOLEAN IWireFrame(BOOLEAN isSizable      = FALSE,  
                           BOOLEAN isDraggable    = FALSE,  
                           UNITS theMinimalWidth  = 0,  
                           UNITS theMinimalHeight = 0);
```

The initializer. The Boolean parameter `isSizable` sets whether the wire frame can be sized. If it is set to `FALSE`, then the wire frame can be used only for dragging purposes. `isDraggable` sets whether the wire frame can be dragged (`TRUE`) or whether it is affixed to one location (`FALSE`). If you do not want the wire frame ever to become smaller than a certain width or height, you can set the dimensions in pixels through `theMinimalWidth` and/or `theMinimalHeight`.

Dragging and Stretching Methods

Wire frames can generate these events: `WFSelectCmd`, `WFDeselectCmd`, and `WFMoveCmd`, depending on whether a view is selected, deselected, or moved/sized. `DoCommands` can pass along data in the form of a void pointer so that wire frame events can send along a pointer to the view object that is selected, deselected, or moved. Thus, users can trap the selection, deselection, or moving of views as these events happen by putting a case in the `DoCommands`.

```
virtual void SetDragging(BOOLEAN isDraggable);
```

An overridden method that is called by the wire frame's owner when it receives a dragging event. The view calls `SetDragging` with a `TRUE` or `FALSE` value.

```
virtual void SetSizing(BOOLEAN isSizable);
```

An overridden method that is called by the wire frame's owner when it receives a sizing event. The view calls `SetSizing` with a `TRUE` or `FALSE` value.

```
virtual BOOLEAN IsSizable(void) const;
```

Returns a Boolean value indicating whether the wire frame can receive sizing events.

```
virtual BOOLEAN IsDraggable(void) const;
```

Returns a Boolean value indicating whether the wire frame can receive dragging events.

Selection Methods

```
virtual BOOLEAN IsSelected(void) const;
```

Returns a Boolean value indicating whether the wire frame is currently selected.

```
virtual void Select(void);
```

Selects (draws) a wire frame.

```
virtual void DoDeselect(CView* theView);
```

Deselects all selected wire frames.

```
virtual void Deselect(void);
```

Deselects ("undraws") a wire frame.

```
virtual CView* GetOwner(void) const;
```

Returns a pointer to the CView object that is the wire frame's owner.

```
virtual CSubview* GetGroupEnclosure(void) const;
```

Returns the subview that acts as the enclosure of the group containing this wire frame. Only views sharing the same enclosure can be multiply selected into a group. Thus, two views selected in two separate windows are not part of the same group.

```
virtual void SetGroupEnclosure(CSubview* theGroupEnclosure);
```

In future releases of XVT-Power++, this method may allow users to set the enclosure of a group dynamically. Currently, this method is not supported.

Mouse Events

The mouse methods in this section have been overridden to implement the sizing and dragging of the wire frame. Each of the following methods has a theButton argument that specifies which mouse button is used and can have the values 0 (left), 1 (middle), or 2 (right). By default, neither the SHIFT key nor the CONTROL key is used in conjunction with the mouse button.

```
virtual voidMouseDown(CPoint theLocation,  
                      short theButton    = 0,
```

```

        BOOLEAN isShiftKey = FALSE,
        BOOLEAN isControlKey= FALSE);

```

When the user presses down a mouse button over a wire frame containing local coordinate `theLocation`, the wire frame receives and handles this event. The wire frame responds to a mouse down event by selecting its owner and drawing a rubberband frame around it. The user can select multiple objects by pressing the SHIFT key or clicking the right mouse button.

```

virtual void MouseMove(CPoint theLocation,
                      short theButton = 0,
                      BOOLEAN isShiftKey = FALSE,
                      BOOLEAN isControlKey= FALSE);

```

When the user moves the mouse over a view containing local coordinate `theLocation`, the view receives and handles this event. If the wire frame is in a selected state, it responds to the mouse move event by resizing or dragging the rubberband frame.

```

virtual void MouseUp(CPoint theLocation,
                    short theButton = 0,
                    BOOLEAN isShiftKey = FALSE,
                    BOOLEAN isControlKey= FALSE);

```

When the user releases a mouse button over a view containing local coordinate `theLocation`, the view receives and handles this event. If the wire frame is in a selected state, it responds to the mouse up event by resizing or moving its owner.

Inherited Utility Methods

```

virtual void Draw(const CRect&
                 theClippingRegion);

```

Takes care of any drawing the wire frame must do. `theClippingRegion` is the portion of the wire frame that needs to be drawn, and it is represented in global, window-relative coordinates. If it is set to NULL, the entire wire frame is drawn.

```

virtual void Size(const CRect& theNewSize);

```

An overridden method that sizes the wire frame according to the coordinates of `theNewSize`.

```

virtual void SetOrigin(const CPoint&
                     theDeltaPoint);

```

An overridden method that takes the current origin of the view and adds `theDeltaPoint` to it. Suppose, for example, that a view has an origin of 10,20 and you want to shift the origin by

three pixels to the right. You set the origin by giving it a delta point of 3,0, which shifts the view three pixels horizontally.

```
virtual void SetFont(const FONT& theFont,  
    BOOLEAN isUpdate = FALSE);
```

Takes a font (theFont) and sets the wire frame's environment to have that font. The isUpdate parameter indicates whether this SetFont message is simply an update message or whether it is a "real" SetFont message meaning that this particular view should set its own font to the new font.

Method for Dragging or Sizing the Owner CView Object

```
BOOLEAN SizeOwner(void);
```

Sizes the owner of the wire frame (that is, the view associated with it) according to the wire frame's current dimensions and location. This method is called automatically when a wire frame has been dragged to a new location or resized.

Protected Methods

```
virtual void DrawWireFrame(BOOLEAN isDrawing);
```

Draws the wire frame.

```
virtual void DrawFrameGrabbers(BOOLEAN  
    isDrawing);
```

Draws the handles that appear on the wire frame when it is selected. Dragging these handles with the mouse enables you to resize the wire frame's owner.

Methods for Dragging or Sizing the Wire Frame During Mouse Dragging

```
virtual void Drag(CPoint theNewLocation);
```

Takes a local coordinate, which is the new location for the wire frame, erases the original wire frame, and draws a new wire frame at that coordinate.

```
virtual void ReSize(CPoint theNewLocation);
```

Takes a new point location (a local coordinate) and stretches the wire frame so that it encompasses this point.

Methods for Drawing/Erasing a Group Wire Frame

```
void SetGroupRegion(void);
```

Sets the wire frame drawing to "group" mode, meaning that more than one wire frame has been selected. This method calculates the size of the group wire frame and redraws the wire frame to match that size.


```
void ClearGroupRegion(void);
```

Sets the wire frame drawing from “group” mode to single mode, meaning that only one wire frame is now selected. In other words, it is the opposite of SetGroupRegion.

Methods for Checking Whether a Mouse Click is for Dragging or Sizing

```
virtual BOOLEAN DoDraggingCheck(const CPoint& theLocation);
```

Given a point, checks to see whether this point is in the dragging area of the wire frame.

```
virtual BOOLEAN DoBorderCheck(const CPoint& theLocation);
```

Given a point, checks to see whether this point is in the sizing area of the wire frame.

Private Methods

Scrolling Event Methods

```
virtual void DoAutoScroll(UNITS theHorizontalIncrement, UNITS theVerticalIncrement);
```

Overrides the AutoScroll method on CView. This method automatically scrolls the wire frame if it is inside a scroller or other view that can be autoscrolled. theHorizontalChange takes a number of pixels. If the number is zero (0), no autoscrolling occurs; if the number is positive, the view scrolls to the right; if it is negative, the view scrolls to the left. theVerticalChange also takes a number of pixels. If the number is zero (0), no autoscrolling occurs; if the number is positive, the view scrolls downward; if it is negative, the view scrolls upward.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

```
virtual void UpdateUnits(CUnits* theUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void Disable(void);
virtual void DoActivate(void);
virtual void DoCommand(long theCommand,void* theData=NULL);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE,BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE,BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
```

```
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjs(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void Hide(void);
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theID);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void Show(void);
```

```
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

Summary: CWireFrame.h

```
#ifndef CWireFrame_H
#define CWireFrame_H

#include "CView.h"

#define UPDATE 0x1234

typedef enum
{
    DRAGGING, SIZING, IDLE, SELECTED, SCROLLING
}
STATE;

typedef enum
{
    WFSHOW, WFHIDE, WFUPDATE
}
DRAWSTATE;

class CWindow;

class CWireFrame : public CView
{
public:
    static const int WIRESIZE; // the size difference with itsOwner

    // constructor && destructor && init
    CWireFrame(CView *theOwner, CSubview* theGroupEnclosure = NULL);
    CWireFrame(const CWireFrame& theFrame);
    CWireFrame& operator=(const CWireFrame& theFrame);
    virtual ~CWireFrame(void);

    virtual BOOLEAN IWireFrame(BOOLEAN isSizable    = FALSE,
                                BOOLEAN isDraggable  = FALSE,
                                UNITS theMinimalWidth = 0,
                                UNITS theMinimalHeight = 0);

    // utility function
    virtual void Draw(const CRect& theClippingRegion);

    // Dragging and stretching functions:
    virtual void SetDragging(BOOLEAN isDraggable);
    virtual void SetSizing(BOOLEAN isSizable);

    virtual BOOLEAN IsSizable(void) const;
    virtual BOOLEAN IsDraggable(void) const;

    virtual BOOLEAN IsSelected(void) const;
    virtual void Select(void);

    virtual void DoDeselect(CView* theView);
    virtual void Deselect(void);

    virtual CView* GetOwner(void) const;

    virtual CSubview* GetGroupEnclosure(void) const;
    virtual void SetGroupEnclosure(CSubview* theGroupEnclosure);
```

```

// mouse events
virtual void MouseDown(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0,
    BOOLEAN isShiftKey=FALSE,
    BOOLEAN isControlKey=FALSE);

//Inherited utility:
virtual void Size(const CRect& theNewSize);
virtual void SetOrigin(const CPoint& theDeltaPoint);
virtual void SetFont(const FONT& theFont, BOOLEAN isUpdate = FALSE);

//Drag or size the owner CView object;
BOOLEAN SizeOwner(void);

protected:

    STATE itsState;           // my current event state
    DRAWSTATE itsDrawState;   // internal drawing state flag
    CPoint itsGrabPoint;      // the point that was grabbed

    long itsSelectCommand;    // command generated upon selection
    long itsDeselectCommand;  // command generated upon deselection
    long itsSizeCommand;      // command generated upon moving/sizing

// Draw the wireframe elements:
virtual void DrawWireFrame(DRAWSTATE isDrawing);
virtual void DrawFrameGrabbers(DRAWSTATE isDrawing);

// Drag and size the wireframe during mouse dragging:
virtual void Drag(CPoint theNewLocation);
virtual void ReSize(CPoint theNewLocation);

// Check if click is for dragging or sizing:
virtual BOOLEAN DoDraggingCheck(const CPoint& theLocation);
virtual BOOLEAN DoBorderCheck(const CPoint& theLocation);

private:
    long itsSizeSide;         // the side which is sizing
    CRect itsMinimalRegion;   // the smallest size possible
    BOOLEAN itIsDraggable;    // can I recieve dragging events
    BOOLEAN itIsSizable;      // can I recieve sizing events
    PNT* itsFramePnts;        // the ending point of the sizing area
    CView* itsOwner;          // the CView it sizes
    BOOLEAN itsGrabState;     // internal grabber state flag

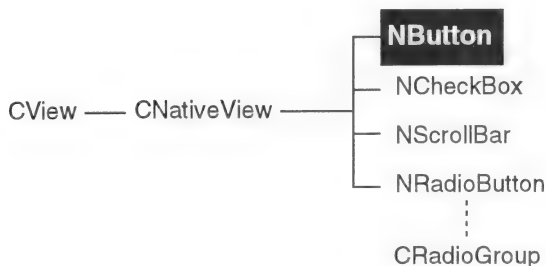
    BOOLEAN isGroupMode;      // is the wire frame in group mode?
    UNITS itsGroupLeft, itsGroupRight, itsGroupTop, itsGroupBottom;

// Scrolling events:
virtual void DoAutoScroll(UNITS theHorizontalIncrement,
    UNITS theVerticalIncrement);
};

#endif CWireFrame_H

```

NButton



Description

NButton objects are push buttons that have the look and feel of buttons of the native window manager.

Heritage

Superclass: CNativeView

Usage

Create and initialize an NButton object with an enclosing CView. When the user clicks on a button, it generates a DoCommand (itsCommand) event.

Data Members

None.

Public Methods

Constructor, Destructor, and Initializer Methods

```

NButton(CSubview* theEnclosure,
        const CRect& theRegion,
        const CString theTitle = NULLString,
        long theControlAttributes = NULL);
  
```

A constructor. theEnclosure is a pointer to the subview that will contain the button. theRegion is a coordinate location, local to the enclosure, that is used to place the button. theTitle is a character string that is used as the title for the button. theControlAttributes takes a value from a set of XVT Portability Toolkit-provided attributes that you can give to native views. You can OR together the appropriate control flag constants into an attribute value. Refer to the table in the section

on CNativeView for a listing of the possible control flags for theControlAttributes. This table appears in the description of the CNativeView constructor.

NButton(const NButton& theButton);

A copy constructor. It duplicates the attributes of a button.

NButton& operator=(const NButton& theButton);

An assignment operator. It duplicates the attributes of a button.

virtual ~NButton(void);

The destructor, which cleans up after the button.

**BOOLEAN IButton(const CString theTitle= NULLString,
 BOOLEAN isEnabled = TRUE,
 long theCommand = NULLcmd,
 BOOLEAN isVisible = TRUE,
 GLUETYPE theGlue = NULLSTICKY);**

The initializer. It takes a title and a Boolean value specifying whether the button is enabled to receive events.theCommand is the command that is generated when the button icon is pressed. Finally, this initializer takes a visibility state and a glue type.

Mouse Hit Method

**virtual void DoHit(CONTROL_INFO
 theControlInfo);**

The button's event-receiving method. It takes a parameter of type CONTROL_INFO, which is defined by the XVT Portability Toolkit in the *XVT Programmer's Guide*. The button responds by sending a DoCommand message.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

virtual void ChangeFont(FONT theFont,FONT_PART theChange);

virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);

virtual CUnits* GetUnits(void) const;

virtual void SetUnits (CUnits* theCoordinateUnits);

From CView

virtual void Activate(void);

virtual void Deactivate(void);

```

virtual void DoActivate(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual long GetDoubleCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;

```



```

virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjs(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CNativeView

```

virtual void Disable(void);
virtual void Enable(void);
WIN_TYPE GetXVTType(void) const;
WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview *thenclature);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theId);
virtual void SetOrigin(const CPoint& diff);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void UpdateUnits(CUnits* theUnits);

```

Summary: NButton.h

```

#ifndef NButton_H
#define NButton_H

#ifdef STDH
#include "CNativeView.h"
#endif STDH
#ifdef DOS
#include "CNativVw.h"
#endif DOS

class NButton : public CNativeView
{
public:
    // constructor && destructor && init
    NButton(CSubview* theEnclosure, const CRect& theRegion,
            const CString theTitle = NULLString,
            long theAttributes = NULL);

    NButton(const NButton& theButton);
    NButton& operator=(const NButton& theButton);

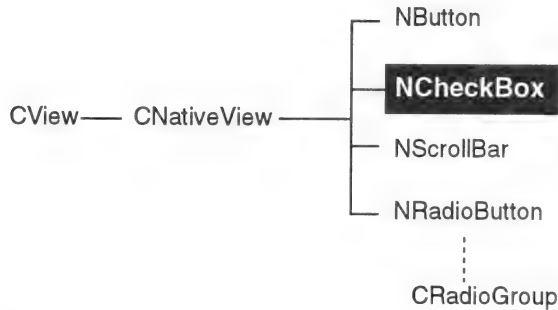
    BOOLEAN IButton(const CString theTitle = NULLString,
                    BOOLEAN isEnabled = TRUE,
                    long theCommand = NULLcmd,
                    BOOLEAN isVisible = TRUE,
                    GLUETYPE theGlue = NULLSTICKY);

    // Mouse hit
    virtual void DoHit(CONTROL_INFO theControlInfo);
};

#endif NButton_H

```

NCheckBox



Description

NCheckBox provides check box objects that have the look and feel of check boxes in the native window manager. A check box is a box with a title beside it. When the user clicks on the box, it becomes checked (selected); upon another click, it becomes deselected, and the check mark (or other indicator of a “checked” status) disappears.

Heritage

Superclass: CNativeView

Usage

Instantiate an NCheckBox object and initialize it. Upon being selected or deselected, the object generates a DoCommand event.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

BOOLEAN	itIsSelected;	selection state
long	itsSelectCommand;	command produced at select time
long	itsDeselectCommand;	command produced at deselect time

Public Methods

```
NCheckBox(CSubview* theEnclosure, const CRect& theRegion,
          const CString theTitle           = NULLString,
          long theControlAttributes        = NULL);
```

A constructor. theEnclosure is a pointer to the subview that will contain the check box. theRegion is a coordinate location, local to the enclosure, that is used to place the check box. Actually, only the top left portion of the region is used, and the width and height of the check box depend on the title of the control and the size definitions of the platform. This constructor also takes a title. theControlAttributes takes a value from a set of XVT Portability Toolkit-provided attributes that you can give to native views. You can OR together the appropriate control flag constants into an attribute value. Refer to the table in the section on CNativeView for a listing of the possible control flags for theControlAttributes. This table appears in the description of the CNativeView constructor.

```
NCheckBox(const NCheckBox& theCheckBox);
```

A copy constructor. It duplicates the attributes of a check box.

```
NCheckBox& operator=(const NCheckBox&
                     theCheckBox);
```

An assignment operator. It duplicates the attributes of a check box.

```
BOOLEAN ICheckBox(const CString& theTitle= NULLString,
                  BOOLEAN isEnabled    = TRUE,
                  long theDeselectCommand = NULLcmd,
                  long theSelectCommand  = NULLcmd,
                  BOOLEAN isVisible     = TRUE,
                  long theGlue          = NULLSTICKY);
```

The initializer. It takes a title, a Boolean value specifying whether the check box is enabled to receive events, a visibility state, and a glue type. In addition it takes two parameters that are special to it: theDeselectCommand is the command that is generated when the check box changes from a selected to a deselected state. Conversely, theSelectCommand is generated when the check box changes from a deselected to a selected state.

Check Box Utilities

```
virtual BOOLEAN IsSelected(void) const;
```

Returns a Boolean value indicating whether the check box is in a selected state (TRUE) or a deselected state (FALSE).

virtual void Select(void);

Selects a check box.

virtual void Deselect(void);

Deselects a check box.

virtual long GetSelectCommand(void) const;

Returns the number of the command that is generated when the check box becomes selected.

**virtual void SetSelectCommand(long
theNewSelectCommand);**

Sets the command that is generated when the check box becomes selected.

virtual long GetDeselectCommand(void) const;

Returns the number of the command that is generated when the check box becomes deselected.

**virtual void SetDeselectCommand(long
theNewSelectCommand);**

Sets the command that is generated when the check box becomes deselected.

Inherited Methods

**virtual void DoHit(CONTROL_INFO
theControlInfo);**

The check box's event-receiving method. It takes a parameter of type CONTROL_INFO, which is defined by the XVT Portability Toolkit in the *XVT Programmer's Guide*. The check box responds by changing to the appropriate selection state and generating the appropriate command for indicating its new state.

virtual void SetSizing(BOOLEAN isSizable);

Takes a Boolean value that sets the sizing of a check box to TRUE so that the check box can be sized or to FALSE so that it cannot be sized. Currently, check boxes cannot be sized. It is a nonportable behavior to size check boxes, which are usually a fixed size according to the native look and feel of the user platform.

**virtual void SetTitle(const CString&
theNewTitle);**

Gives the check box the title designated by theNewTitle.

Protected Methods

None.

Private Methods

void UpdateSize(void);

An internal method that is called to size the check box after its title changes.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

virtual void ChangeFont(FONT theFont, FONT_PART theChange);

virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);

virtual CUnits* GetUnits(void) const;

virtual void SetUnits (CUnits* theCoordinateUnits);

From CView

virtual void Activate(void);

virtual void Deactivate(void);

virtual void DoActivate(void);

virtual void DoCommand(long theCommand, void* theData=NULL);

virtual void DoDeactivate(void);

virtual void DoDisable(void);

virtual void DoDraw(void);

virtual void DoDraw(const CRect& theClippingRegion);

virtual void DoEnable(void);

virtual void DoHide(void);

virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);

virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoMouseUp(CPoint theLocation, short theButton = 0, BOOLEAN isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void DoSetDragging(BOOLEAN isDraggable);

```
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
```

```

virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CNativeView

```

virtual void Disable(void);
virtual void Enable(void);
WIN_TYPE GetXVTType(void) const;
WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview *thenclosure);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetOrigin(const CPoint& diff);
virtual void UpdateUnits(CUnits* theUnits);

```

Summary: NCheckBox.h

```

#ifndef NCheckBox_H
#define NCheckBox_H

#ifdef STDH
#include "CNativeView.h"
#endif STDH
#ifdef DOS
#include "CNativVw.h"
#endif DOS

```



```
class NCheckBox : public CNativeView
{
    public:
        NCheckBox(CSubview* theEnclosure, const CRect& theRegion,
            const CString theTitle = NULLString,
            long theControlAttributes = NULL);
        NCheckBox(const NCheckBox& theCheckBox);
        NCheckBox& operator=(const NCheckBox& theCheckBox);

        BOOLEAN ICheckBox(const CString& theTitle,
            BOOLEAN isEnabled = TRUE,
            long theDeselectCommand = NULLcmd,
            long theSelectCommand = NULLcmd,
            BOOLEAN isVisible = TRUE,
            long theGlue = NULLSTICKY);

        // Check box utilities:
        virtual BOOLEAN IsSelected(void) const;
        virtual void Select(void);
        virtual void Deselect(void);

        virtual long GetSelectCommand(void) const;
        virtual void SetSelectCommand(long theNewSelectCommand);

        virtual long GetDeselectCommand(void) const;
        virtual void SetDeselectCommand(long theNewSelectCommand);

        // Inherited methods:
        virtual void DoHit(CONTROL_INFO theControlInfo);
        virtual void SetSizing(BOOLEAN isSizable);
        virtual void SetTitle(const CString& theNewTitle);

    private:
        BOOLEAN itIsSelected; // selection state
        long itsSelectCommand; // command produced at select time
        long itsDeselectCommand; // command produced at deselect time
};

#endif NCheckBox_H
```

NLineText



Description

NLineText provides a single-line text editing box inside of a CView. It is commonly used for one-line text entries, such as a place for users to enter their names or passwords. You must specify the length; the height is calculated from the font of the text.

Heritage

Superclass: CNativeTextEdit

Usage

Instantiate an NLineText object and initialize its length.

Environment

The text is drawn in the foreground color; the border is drawn with the *pen* color. The interior of the text box is drawn in the background color.

Data Members

None.

Public Methods

Constructor, Destructor, and Initializer Methods

```

NLineText(CSubview* theEnclosure,
          const CPoint& theTopLeftPoint,
          UNITS theLength,
          unsigned theAttributes = TX_BORDER,
          UNITS theRightMargin = 1000,
          int theCharacterLimit = 1000);
  
```

The constructor. It takes an enclosure, which is the view that will contain the NLineText object. theTopLeftPoint specifies the coordinate at which the line of text starts, and theLength indicates the length of the line. The height of the line is determined by the size of the font (in pixels). By default, the NLineText object has a border, as determined by the XVT value

given to theAttributes. Finally, this constructor takes a right margin for the text line, which is measured in pixels starting with zero (0) at the left side of the text box, and a limit on the number of characters that can fit into the text line.

```
NLineText(const NLineText& theLineText);
```

A copy constructor that duplicates the attributes of an NLineText object, as well as the text inside it.

```
NLineText& operator=(const NLineText& theLineText);
```

An assignment operator that duplicates the attributes of an NLineText object, as well as the text inside it.

```
BOOLEAN ILineText(unsigned theAttributes      = TX_BORDER,  
                   UNITS theRightMargin      = 1000,  
                   int theCharactedLimit      = 1000,  
                   const CString theInitialText= NULLString,  
                   BOOLEAN isAutoSelected     = FALSE,  
                   BOOLEAN isVisible          = TRUE,  
                   long theGlue                = SAMESTICKY);
```

The initializer. It is identical to the initializer of CNativeTextEdit. It takes an unsigned value pertaining to the XVT attributes that the text edit system can have. These attributes can be ORED and passed in together, as described in the text editing chapter of the *XVT Programmer's Guide*. For further information about the possible values of theAttributes, see the table in the section on CNativeTextEdit, under INativeTextEdit.

The next parameter, theRightMargin is a pixel measurement of the right margin, starting from the left side and beginning at zero (0). In addition, there is a parameter that sets the default character limit at 1,000 characters. theInitialText is a string containing any text you may wish to initialize with the text edit system. If the isAutoSelected parameter is set to TRUE, all of the text in the text edit box becomes selected when the user clicks inside the box. Finally, this initializer, like the initializers of all CView classes, takes a visibility state and a glue type.

Inherited Utility

```
virtual void SetAttribute(unsigned  
theAttribute, BOOLEAN isSet = TRUE);
```

Sets an attribute for a text box. You specify the attribute and give it a Boolean value indicating whether to turn this attribute on or off. See INativeTextEdit for further information about

XVT attributes. Also, see the text editing chapter of the *XVT Programmer's Guide*.

```
virtual void SetFont(const FONT& aFont,  
    BOOLEAN isUpdate = FALSE);
```

Takes a font and sets the text box's environment to have that font. The `isUpdate` parameter indicates whether this `SetFont` message is simply an update message or whether it is a "real" `SetFont` message meaning that the text box should set its own font to the new font.

```
virtual void Size(const CRect& theNewSize);
```

Resets the region of the text box. Possibly the region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, `theNewSize`, like the coordinates of the region that is passed in on creation of a text box, are relative to the enclosure. Changes in height are ignored since these changes depend on the object's font.

Protected Methods

None.

Private Methods

```
int ResetHeight(const FONT& theFont, const  
    CWindow& theWindow) const;
```

An internal method that is called when the font of the text object is changed and the height of the text object must therefore be adjusted.

```
CRect GetInitialFrame(CSubview* theEnclosure,  
    const CPoint& theTopLeft, UNITS theLength);
```

An internal method that calculates the initial size of the object during construction.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
    isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Deactivate(void);
virtual void DoActivate(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual long GetDoubleCommand(void) const;
virtual CWindow* GetCWindow(void) const;
```

```
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void MouseMove(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetId(int theID);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

From CNativeTextEdit

```

virtual void Activate(void);
virtual BOOLEAN Append(const CString& theText);
virtual BOOLEAN Clear(void);
virtual void Disable(void);
virtual void Draw(const CRect& theClipping);
virtual unsigned long GetAttributes(void) const;
void GetFullPar(T_PNUM theParagraph, CString& theText) const;
void GetLineInternal(T_PNUM theParagraph, T_LNUM theLine,   CString& theText) const;
virtual int GetMargin(void) const;
virtual T_CNUM GetNCharInSelection(void) const;
T_CNUM GetNCharInternal(T_PNUM theFirstParagraph, T_PNUM theLastParagraph, T_LNUM
theFirstLine, T_LNUM theLastLine,   T_CNUM theFirstChar, T_CNUM theLastChar) const;
virtual T_CNUM GetNCharInText(void) const;
virtual int GetLimit(void) const;
void GetPartLine(T_CNUM theFirstChar, T_CNUM theLastChar, T_PNUM
theParagraph, T_LNUM theLineNumber, CString& theText) const;
void GetPartPar(T_CNUM theStartChar, T_CNUM theEndChar, T_PNUM theParagraph, T_LNUM
theStartLine, T_LNUM theEndLine, CString& theText) const;
virtual CString GetSelectedText(void) const;
virtual CString GetText(void) const;
void GetTextInternal(T_PNUM theFirstParagraph, T_PNUM theLastParagraph, T_LNUM
theFirstLine, T_LNUM theLastLine,   T_CNUM theFirstChar, T_CNUM theLastChar,
CString& theTextBuffer) const;
virtual void Hide(void);
BOOLEAN IsEmpty(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton=0, BOOLEAN isShiftKey
= FALSE, BOOLEAN isControlKey = FALSE);
virtual void MouseDown(CPoint theLocation, short theButton=0,   BOOLEAN isShiftKey =
FALSE, BOOLEAN isControlKey = FALSE);
virtual void SelectText(void);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetLimit(int theCharacterLimit);
virtual void SetMargin(int theRightMargin);
virtual void SetOrigin(const CPoint& theDelta);
virtual void SetText(const CString& theText);
virtual void Show(void);

```

```
void Truncate(CString& theText);
virtual void UpdateUnits(CUnits* theUnits);
virtual int Validate(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
```

Summary: NLineText.h

```
#ifndef NLineText_H
#define NLineText_H

#ifdef STDH
#include "CNativeTextEdit.h"
#endif STDH
#ifdef DOS
#include "CNTvTxE.h"
#endif DOS

class NLineText : public CNativeTextEdit
{
public:
    // construc & destruct & init
    NLineText(CSubview* theEnclosure, const CPoint& theTopLeftPoint,
        UNITS theLength,
        unsigned theAttributes          = TX_BORDER,
        UNITS theRightMargin            = 1000,
        int theCharacterLimit           = 1000);
    NLineText(const NLineText& theLineText);
    NLineText& operator=(const NLineText& theLineText);

    BOOLEAN ILineText(unsigned theAttributes = TX_BORDER,
        UNITS theRightMargin = 1000,
        int theCharactedLimit = 1000,
        const CString theInitialText = NULLString,
        BOOLEAN isAutoSelector = FALSE,
        BOOLEAN isVisible = TRUE,
        long theGlue = SAMESTICKY);

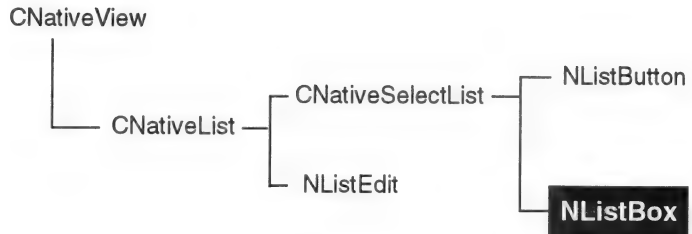
    // Inherited utility:
    virtual void SetAttribute(unsigned theAttribute,
        BOOLEAN isSet = TRUE);

    virtual void SetFont(const FONT& aFont, BOOLEAN isUpdate = FALSE);
    virtual void Size(const CRect& theNewSize);

private:
    int ResetHeight(const FONT& theFont,
        const CWindow theWindow) const;
    CRect GetInitialFrame(CSubview* theEnclosure,
        const CPoint& theTopLeft, UNITS theLength);
};

#endif NLineText_H
```


NListBox



Description

NListBox is a class that is based on XVT's native list box control, which allows a user to select one or more items from a scrollable list. In contrast to CListBox, which has an XVT-Power++ look and feel, NListBox has the look and feel of the native window manager.

Heritage

Superclass: CNativeSelectList

Subclasses: None

Usage

Create an instance of this class and initialize it as you would any other native view.

Environment

There are no environment settings for classes that inherit from CNativeView. The drawing properties of native views are system-defined.

Data Members

None.

Public Methods

NListBox(CSubview* theEnclosure, const CRect& theRegion, const CStringList& theItems, long theControlAttributes = CTL_FLAG_MULTIPLE);

A constructor. theEnclosure is a pointer to the subview that will contain the list box. theRegion is a coordinate location, local to the enclosure, that is used to place the native list.

theItems is a list of the strings that the native select list will contain. TheControlAttributes specifies the XVT attributes that determine the special characteristics and initial state of a control when it is created. The possible XVT control flags vary from control to control, and only a couple of them are generic to all controls. For information on the possible attributes a particular type of control can have, see the discussions of create_control and CTL_FLAG in the *XVT Programmer's Reference*. Note that this method takes no title and that the function SetTitle has no effect on this object. It has no title because the list box is simply a list of selections.

BOOLEAN IListBox(const CStringList& theItems);

Clears any items out of the list box and inserts the list of strings (theItems) into it. It returns a Boolean value of TRUE if it succeeds and FALSE if it fails.

NListBox(const NListBox& theNativeList);

A copy constructor. It duplicates the attributes of a native list box. The items inside the list box are currently not being copied.

NListBox& operator=(const NListBox& theNativeList);

An assignment operator. It duplicates the attributes of a native list box. The items inside the list box are currently not being copied.

Inherited Methods

virtual void DoHit(CONTROL_INFO theControlInfo);

The primary event-receiving method for native list boxes. It takes a parameter of type CONTROL_INFO, which is defined by XVT in the *XVT Programmer's Guide*. When the user clicks, it passes a click command, and when the user double clicks, it passes a double click command—all through the DoCommand mechanism. This method not only gets the click command but also the object that is clicked or double clicked. In short, DoHit translates single and double clicks through the DoCommand mechanism.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);  
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);  
virtual CUnits* GetUnits(void) const;  
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Activate(void);  
virtual void Deactivate(void);  
virtual void DoActivate(void);  
virtual void DoCommand(long theCommand, void* theData=NULL);  
virtual void DoDeactivate(void);  
virtual void DoDisable(void);  
virtual void DoDraw(void);  
virtual void DoDraw(const CRect& theClippingRegion);  
virtual void DoEnable(void);  
virtual void DoHide(void);  
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);  
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoSetDragging(BOOLEAN isDraggable);  
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);  
virtual void DoSetGlue(GLUETYPE theGlue);  
virtual void DoSetOrigin(const CPoint& theDeltaPoint);  
virtual void DoSetSizing(BOOLEAN isSizable);  
virtual void DoShow(void);  
virtual void DoSize(const CRect& theNewSize);  
virtual void DoTimer(long theTimerId);  
virtual void DoUser(long theUserId, void* theData);  
virtual void Draw(void);  
virtual void Draw(const CRect& theClippingRegion);
```

```

virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

```

```

void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CNativeView

```

virtual void Disable(void);
virtual void Enable(void);
WIN_TYPE GetXVTType(void) const;
WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview *thenclature);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theId);
virtual void SetOrigin(const CPoint& diff);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void UpdateUnits(CUnits* theUnits);

```

From CNative List

```

virtual BOOLEAN Clear(void);
virtual CStringList GetAllItems(void);
virtual CString GetItem(int thePosition,
    int theNumberOfCharacters=ITEMLENGTH);
virtual int GetNumItems(void);
virtual BOOLEAN Insert(const CStringList& theStringList, int thePosition=-1);
virtual BOOLEAN Insert(const CString& theString, int thePosition=-1);
virtual BOOLEAN Remove(int thePosition);

```

Summary: NListBox.h

```

#ifndef NListBox_H
#define NListBox_H

```

```
#include "PwrNames.h"
#include "PwrDef.h"

#include CNativeSelectList_i

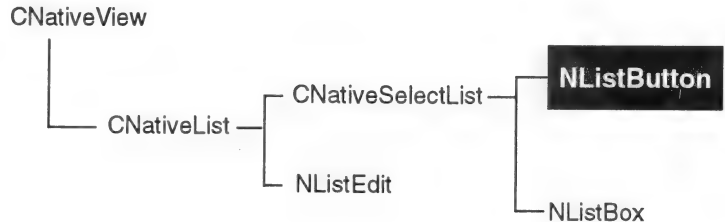
class CString;
class CStringList;

class NListBox : public CNativeSelectList
{
public:
    NListBox(CSubview* theEnclosure,
             const CRect& theRegion,
             const CStringList& theItems,
             long theControlAttributes = CTL_FLAG_MULTIPLE);

    BOOLEAN IListBox(const CStringList& theItems);
    NListBox(const NListBox& theNativeList);
    NListBox& operator=(const NListBox& theNativeList);
    // Inherited:
    virtual void DoHit(CONTROL_INFO theControlInfo);
};

#endif // NListBox_H
```

NListButton



Description

NListButton is a class that is based on XVT's native list box control, which allows a user to select one or more items from a scrollable list. In contrast to CListBox, which has an XVT-Power++ look and feel, NListBox has the look and feel of the native window manager.

Heritage

Superclass: CNativeSelectList

Subclasses: None

Usage

Create an instance of this class and initialize it as you would any other native view.

Environment

There are no environment settings for classes that inherit from CNativeView. The drawing properties of native views are system-defined.

Data Members

None.

Public Methods

```
NListButton(CSubview* theEnclosure, const  
            CRect& theRegion, const CStringList&  
            theItems, int thePosition = 0, long
```

```
theControlAttributes =  
CTL_FLAG_NATIVE_JUST);
```

A constructor. `theEnclosure` is a pointer to the subview that will contain the list button. `theRegion` is a coordinate location, local to the enclosure, that is used to place the list button. `theItems` is a list of the strings that the pop-up list box will contain. `thePosition` is the line number of the text item that will serve as the label of the button when it is created. `theControlAttributes` specifies the XVT attributes that determine the special characteristics and initial state of a list button when it is created. See the discussions of `create_control` and `CTL_FLAG` in the *XVT Programmer's Reference*. Here, the default is `CTL_FLAG_NATIVE_JUST`, which means that the items in the pop-up list box will have native text justification.

```
BOOLEAN IListButton(const CStringList&  
theItems,int thePosition=0);
```

The initializer. It takes a list of the strings to be contained in the button's pop-up list box (`theItems`) and the line number (`thePosition`) of the text item that will serve as the button's label when the button is created.

```
NListButton(const NListButton& theNativeList);
```

A copy constructor. It duplicates the attributes of a list button. The items inside the list button are currently not being copied.

```
NListButton& operator=(const NListButton&  
theNativeList);
```

An assignment operator. It duplicates the attributes of a list button. The items inside the list button are currently not being copied.

Inherited Methods

```
virtual void DoHit(CONTROL_INFO  
theControlInfo);
```

The primary event-receiving method for native list buttons. It takes a parameter of type `CONTROL_INFO`, which is defined by XVT in the *XVT Programmer's Guide*. When the user clicks on the button, it passes a click command through the `DoCommand` mechanism. This method not only gets the click command but also the button that is clicked. In short, `DoHit` translates mouse clicks through the `DoCommand` mechanism.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);  
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);  
virtual CUnits* GetUnits(void) const;  
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Activate(void);  
virtual void Deactivate(void);  
virtual void DoActivate(void);  
virtual void DoCommand(long theCommand, void* theData=NULL);  
virtual void DoDeactivate(void);  
virtual void DoDisable(void);  
virtual void DoDraw(void);  
virtual void DoDraw(const CRect& theClippingRegion);  
virtual void DoEnable(void);  
virtual void DoHide(void);  
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);  
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoSetDragging(BOOLEAN isDraggable);  
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);  
virtual void DoSetGlue(GLUETYPE theGlue);  
virtual void DoSetOrigin(const CPoint& theDeltaPoint);  
virtual void DoSetSizing(BOOLEAN isSizable);  
virtual void DoShow(void);  
virtual void DoSize(const CRect& theNewSize);  
virtual void DoTimer(long theTimerId);
```

```
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
```

```

virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

void Prepare(const CRect& theClippingRegion);

virtual void SetCommand(long theCommand);

virtual void SetDoubleCommand(long theCommand);

virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);

virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);

virtual void SetWireFrame(CWireFrame* theNewWireFrame);

virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CNativeView

```

virtual void Disable(void);

virtual void Enable(void);

WIN_TYPE GetXVTType(void) const;

WINDOW GetXVTWindow(void) const;

virtual void Hide(void);

virtual void SetDragging(BOOLEAN isDraggable);

virtual void SetEnclosure(CSubview *thenclosure);

virtual void SetGlue(GLUETYPE theGlue);

virtual void SetId(int theId);

virtual void SetOrigin(const CPoint& diff);

virtual void SetSizing(BOOLEAN isSizable);

virtual void SetTitle(const CString& theNewTitle);

virtual void Show(void);

virtual void Size(const CRect& theNewSize);

virtual void UpdateUnits(CUnits* theUnits);

```

From CNative List

```

virtual BOOLEAN Clear(void);

virtual CStringList GetAllItems(void);

virtual CString GetItem(int thePosition,
    int theNumberOfCharacters=ITEMLENGTH);

virtual int GetNumItems(void);

virtual BOOLEAN Insert(const CStringList& theStringList, int thePosition=-1);

virtual BOOLEAN Insert(const CString& theString, int thePosition=-1);

virtual BOOLEAN Remove(int thePosition);

```

Summary: NListButton.h

```
#ifndef NListButton_H
#define NListButton_H

#include "PwrNames.h"
#include "PwrDef.h"

#include CNativeSelectList_i

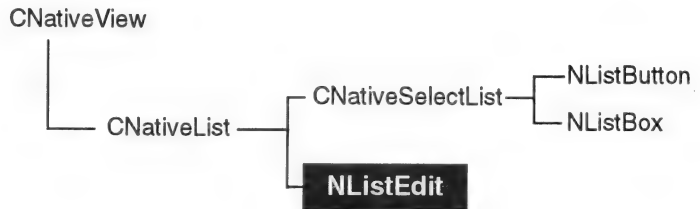
class CString;
class CStringList;

class NListButton : public CNativeSelectList
{
public:
    NListButton(CSubview* theEnclosure,
               const CRect& theRegion,
               const CStringList& theItems,
               int thePosition = 0,
               long theControlAttributes = CTL_FLAG_NATIVE_JUST);

    BOOLEAN IListButton(const CStringList& theItems, int thePosition=0);
    NListButton(const NListButton& theNativeList);
    NListButton& operator=(const NListButton& theNativeList);
    // Inherited:
    virtual void DoHit(CONTROL_INFO theControlInfo);
};

#endif // NListButton_H
```

NListEdit



Description

NListEdit is a class that maps to XVT's list edit. **NListEdit** provides a text field with a small button beside it. Pressing the button invokes a pop-up list box of text items; selecting one of these items and releasing the mouse button causes the selected item to be inserted into the text field.

Heritage

Superclass: **CNativeList**

Subclasses: None

Usage

Create an instance of this class and initialize it as you would any other native view.

Environment

There are no environment settings for classes that inherit from **CNativeView**. The drawing properties of native views are system-defined.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

long	itsKeyFocusCmd;	the command generated when the keyboard gains focus on
------	-----------------	--

long	itsKeyLostFocusCmd;	the edit field the command generated when the keyboard loses focus on the edit field
long	itsTextCmd;	the command generated when the textual contents of the edit field change

Public Methods

Constructor, Destructor, and Initializer Methods

NListEdit(CSubview* theEnclosure, const CRect& theRegion, const CStringList& theItems, const CString& theTitle = NULLString);

A constructor. theEnclosure is a pointer to the subview that will contain the list edit. theRegion is a coordinate location, local to the enclosure, that is used to place the native list. theItems is a list of the strings that the list edit will contain. theTitle is the string that will serve as the title of the native list. This title is the actual item in the edit field that has been selected from the pop-up list of items.

NListEdit(const NListEdit& theNativeList);

A copy constructor. It duplicates the attributes of a list edit. The items inside the list edit are currently not being copied.

BOOLEAN IListEdit(const CStringList& theItems, const CString& theTitle = NULLString);

Clears out any text items present in the list, inserts the strings given to theItems, and sets the title.

NListEdit& operator=(const NListEdit& theNativeList);

An assignment operator. It duplicates the attributes of a list edit. The items inside the list edit are currently not being copied.

Command Settings

virtual void SetFocusCommands(long theKeyFocusCommand, long theKeyLostFocusCommand);

Within the edit field of the list edit, the keyboard can either gain or lose focus, and commands can be generated in either case. This method sets the commands that are generated when the keyboard gains or loses focus. In each case, a command is sent using the normal DoCommand chain of events.

```
virtual void SetTextCommand(long  
theTextCommand);
```

Allows you to set the command that is generated when the textual contents in the edit field change.

```
long GetKeyFocusCommand(void) const;
```

Returns a list edit's keyboard focus command, that is, the command that is generated when the keyboard gains focus on the edit field of the list edit.

```
long GetKeyFocusLostCommand(void) const;
```

Returns the command that is generated when the keyboard loses focus on the edit field of the list edit.

```
long GetTextCommand(void) const;
```

Returns the command that is generated when the textual contents in the edit field change.

Inherited Methods

```
virtual void DoHit(CONTROL_INFO  
theControlInfo);
```

When the button of the list edit receives a mouse event, this method translates the event into one of three commands that are sent through the DoCommand chain: theKeyFocusCommand, theKeyLostFocusCommand or theTextCommand.

```
virtual void Activate(void);
```

Sets the state of the list edit to active. This means that it is the list edit that will currently receive user input through keyboard or mouse events.

```
virtual void Deactivate(void);
```

Sets the state of the list edit to inactive, which means that it is currently not the list edit that will receive user input through keyboard or mouse events.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void DoActivate(void);
virtual void DoCommand(long theCommand,void* theData=NULL);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
```



```

virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CNativeView

```

virtual void Disable(void);
virtual void Enable(void);
WIN_TYPE GetXVTType(void) const;
WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview *thenclature);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theId);
virtual void SetOrigin(const CPoint& diff);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
virtual void UpdateUnits(CUnits* theUnits);

```

From CNative List

```

virtual BOOLEAN Clear(void);
virtual CStringList GetAllItems(void);
virtual CString GetItem(int thePosition,
    int theNumberOfCharacters=ITEMLENGTH);
virtual int GetNumItems(void);
virtual BOOLEAN Insert(const CStringList& theStringList, int thePosition=-1);
virtual BOOLEAN Insert(const CString& theString, int thePosition=-1);
virtual BOOLEAN Remove(int thePosition);

```

Summary: NListEdit.h

```

#ifndef NListEdit_H
#define NListEdit_H

#include "PwrNames.h"
#include "PwrDef.h"

#include CNativeList_i

class CString;
class CStringList;

class NListEdit : public CNativeList
{
public:

```

```
// constructor && destructor && init
NListEdit(CSubview* theEnclosure, const CRect& theRegion,
          const CStringList& theItems,
          const CString& theTitle = NULLString);

NListEdit(const NListEdit& theNativeList);

BOOLEAN IListEdit(const CStringList& theItems,
                  const CString& theTitle = NULLString);

NListEdit& operator=(const NListEdit& theNativeList);

// Command Settings:
virtual void SetFocusCommands(long theKeyFocusCommand,
                              long theKeyLostFocusCommand);
virtual void SetTextCommand(long theTextCommand);

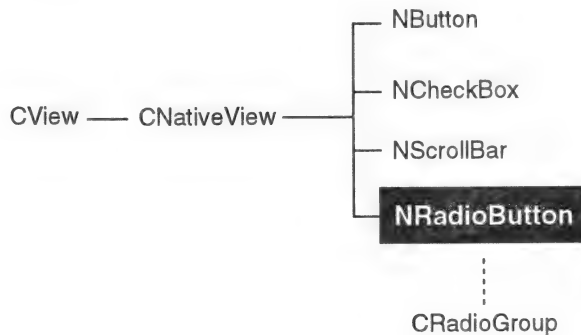
long GetKeyFocusCommand(void) const;
long GetKeyFocusLostCommand(void) const;
long GetTextCommand(void) const;

// Inherited:
virtual void DoHit(CONTROL_INFO theControlInfo);
virtual void Activate(void);
virtual void Deactivate(void);

private:
long itsKeyFocusCmd;
long itsKeyLostFocusCmd;
long itsTextCmd;
};

#endif // CNativeList_H
```

NRadioButton



Description

NRadioButton provides radio button objects (belonging to a group) that have the look and feel of radio buttons in the native window manager. By definition, radio buttons must be grouped together. Unlike check boxes, which can be multiply selected, radio buttons can only be selected one at a time. Selecting one radio button means to deselect another. The helper class **CRadioGroup** serves as a grouper.

Heritage

Superclass: **CNativeView**

Usage

NRadioButton is a protected class that cannot be directly instantiated. Instead, you must create a radio button grouper (**CRadioGroup**) and use either the **AddButton** or **AddButtons** method to create new buttons in the group.

Public Data Members

None.

Protected Data Members

NRadioButton	*itsRadioGroup	a pointer to the group containing the radio button
---------------------	-----------------------	--

Private Data Members

None.

Friends

friend class CRadioGroup;

Public Methods

Inherited Utility Methods

```
virtual void DoHit(CONTROL_INFO controlInfo);
```

The radio button's event-receiving method. It takes a parameter of type CONTROL_INFO, which is defined by XVT in the *XVT Programmer's Guide*. The button responds by informing its group, which takes care of the proper selection. In addition, a DoCommand message is sent.

Sets the sizing of a radio button to TRUE so that the button can be sized or to FALSE so that it cannot be sized. Currently, radio buttons cannot be sized. It is a non-portable behavior to size radio buttons, which are usually a fixed size according to the native look and feel.

```
virtual void SetTitle(const CString&
theNewTitle);
```

Gives the radio button the title designated by theNewTitle. The title of a radio button is the string that appears beside it.

Protected Methods

```
NRadioButton(CSubview *theEnclosure,
             const CRect& theRegion,
             CRadioGroup* theRadioGroup,
             long theAttributes      = NULL,
             const CString& theTitle = NULLString);
```

A constructor. theEnclosure is a pointer to the subview that will contain the radio button. theRegion is a coordinate location, local to the enclosure, that is used to place the radio button. theRadioGroup is a pointer to the group that is to contain the radio button. theAttributes takes a value from a set of XVT-provided attributes that you can give to native views. You can OR together the appropriate control flag constants into an attribute value. Refer to the table in the section on CNativeView for a listing of the possible control flags for theAttributes. This table appears in the description of the CNativeView constructor. Finally, this constructor takes a character string for its title, theTitle.

```
NRadioButton(const NRadioButton& theButton);
```

A copy constructor. It copies the radio button.

```
NRadioButton& operator=(const NRadioButton& theButton);
```

An assignment operator. It copies the radio button.

```
~NRadioButton(void);
```

The destructor. It cleans up after the radio button.

```
BOOLEAN IRadioButton(const CString& theTitle= NULLString,  

                     BOOLEAN isEnabled = TRUE,  

                     long theCommand  = NULLcmd,  

                     BOOLEAN isVisible = TRUE,  

                     long theGlue     = NULLSTICKY);
```

The initializer. It takes a title for the radio button, a Boolean specifying whether the radio button is enabled to receive events, the number of the command that is generated when the button is clicked, a visibility state, and a glue type.

Private Methods

```
void UpdateSize(void);
```

An internal method that updates the size of the radio grouper to encompass any newly added radio buttons.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Activate(void);
```

```
virtual void Deactivate(void);
```

```
virtual void DoActivate(void);
```

```
virtual void DoCommand(long theCommand, void* theData=NULL);
```

```
virtual void DoDeactivate(void);
```

```
virtual void DoDisable(void);
```

```
virtual void DoDraw(void);
```

```
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
```

```

virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CNativeView

```

virtual void Disable(void);
virtual void Enable(void);
WIN_TYPE GetXVTType(void) const;
WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual void SetDragging(BOOLEAN isDraggable);

```



```

virtual void SetEnclosure(CSubview *thenclosure);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetId(int theId);
virtual void SetOrigin(const CPoint& diff);
virtual void UpdateUnits(CUnits* theUnits);

```

Summary: NRadioButton.h

```

#ifndef NRadioButton_H
#define NRadioButton_H

#ifdef STDH
#include "CNativeView.h"
#include "CSubview.h"
#endif
#ifdef DOS
#include "CNativVw.h"
#include "CSubview.h"
#endif

class COrderedList;

class CRadioGroup : public CSubview
{
public:
    CRadioGroup(CSubview *theEnclosure, const CPoint& theTopLeft);
    CRadioGroup(const CRadioGroup& theGroup);
    CRadioGroup& operator=(const CRadioGroup& theGroup);
    virtual ~CRadioGroup(void);

    BOOLEAN IRadioGroup(const CString& theTitle = NULLString,
        BOOLEAN isVisible = TRUE,
        GLUETYPE theGlue = NULLSTICKY);

    // Grouper utility:
    virtual int AddButton(const CPoint& aPoint,
        const CString& theButtonTitle = NULLString,
        long theCommand = NULLcmd,
        long theAttributes = NULL,
        GLUETYPE theGlue = NULLSTICKY);

    virtual int AddButtons(int theFirstResourecId,
        int theNumberOfButtons,
        int theFirstCommand,
        long theAttributes = NULL,
        int theSelectedButton = NULL,
        UNITS theSeparation = 0,
        DIRECTION theDirection = VERTICAL);

    virtual void RemoveButton(int theButtonId);

    virtual void SetSelectedButton(int id);
    virtual int GetSelectedButton(void) const;

    virtual void Draw(const CRect& theClipping);

    //Border space for button placement:
    virtual void SetBorderSpace(int thePixelSpace);
    virtual int GetBorderSpace(void) const;

```

```

private:
    COrderedList *itsButtons; // the list of grouped buttons
    int itsSelectedId;        // is of selected button
    int itsBorderSpace;       // spacing between buttons and grouper

    int AddButtonInternal(const CPoint &aPoint,
        const CString& theButtonText = NULLString,
        long theCommand = NULLCmd,
        long theAttributes = NULL,
        GLUETYPE theGlue = NULLSTICKY);
};

class CRadioButton : public CNativeView
{
    friend class CRadioGroup;

public:
    // Inherited utility:
    virtual void DoHit(CONTROL_INFO controlInfo);
    virtual void SetSizing(BOOLEAN isSizable);
    virtual void SetTitle(const CString& theNewTitle);

protected:
    CRadioGroup *itsRadioGroup;

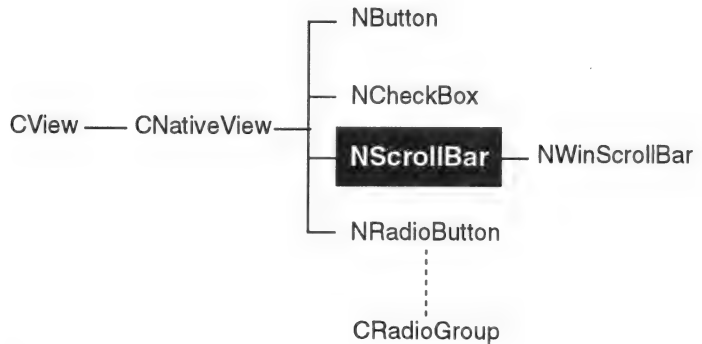
    NRadioButton(CSubview *theEnclosure,
        const CRect& theRegion,
        CRadioGroup* theRadioGroup,
        long theAttributes = NULL,
        const CString& theTitle = NULLString);
    NRadioButton(const CRadioButton& theButton);
    NRadioButton& operator=(const CRadioButton& theButton);
    ~NRadioButton(void);

    BOOLEAN IRadioButton(const CString& theTitle = NULLString,
        BOOLEAN isEnabled = TRUE,
        long theCommand = NULLCmd,
        BOOLEAN isVisible = TRUE,
        long theGlue = NULLSTICKY);
};

#endif NRadioButton_H

```

NScrollBar



Description

NScrollBar objects provide scrollbars that have the look and feel of scrollbars in the native window manager. An NScrollBar object sends HScroll and VScroll (horizontal and vertical scroll) messages to its owner, which must in turn do what is necessary to scroll the objects. By default, a scrollbar's owner is its enclosure.

Heritage

Superclass: CNativeView

Usage

Create an instance of this class and initialize it. The scrollbar updates its thumb automatically and informs its owner of the event.

Public Data Members

None.

Protected Data Members

None.

Private Data Members

UNITS	itsPos;	the scrollbar's position
UNITS	itsMinPosition;	range start
UNITS	itsMaxPosition;	range stop
UNITS	itsLineIncrement;	how much, in pixels, to increment on a line scroll

UNITS	itsPageIncrement;	how much, in pixels, to increment on a page scroll
CView*	itsOwner;	delegate events to owner

Public Methods:

Constructor, Destructor, and Initializer Methods

```
NScrollBar(CSubview* theEnclosure,
            const CRect& theRegion,
            DIRECTION theDirection,
            long theAttributes = NULL);
```

A constructor. theEnclosure is a pointer to the subview that will contain the scrollbar. theRegion is a coordinate location, local to the enclosure, that is used to place the scrollbar. theDirection is an enum that takes two possible values: HORIZONTAL or VERTICAL. theControlAttributes takes a value from a set of XVT-provided attributes that you can give to native views. You can OR together the appropriate control flag constants into an attribute value. Refer to the table in the section on CNativeView for a listing of the possible control flags for theControlAttributes. This table appears in the description of the CNativeView constructor.

```
NScrollBar(const NScrollBar& theScrollBar);
```

A copy constructor. It duplicates the attributes of a scrollbar.

```
NScrollBar& operator=(const NScrollBar&
                      theCScrollbar);
```

An assignment operator. It duplicates the attributes of a scrollbar.

```
BOOLEAN IScrollBar(int theMinPosition    = 0,
                   UNITS theMaxPosition  = 0,
                   UNITS theCurrentPosition = 0,
                   UNITS theLineIncrement = 1,
                   UNITS thePageIncrement = 20,
                   BOOLEAN isEnabled     = TRUE,
                   BOOLEAN isVisible     = TRUE,
                   GLUETYPE theGlue      = NULLSTICKY);
```

The initializer. For a vertical scrollbar, theMinPosition is the top, and theMaxPosition is the bottom. For a horizontal scrollbar, theMinPosition is the left, and theMaxPosition is the right. theCurrentPosition specifies the position of the

thumb, which must be relative to the minimum and maximum positions.

theLineIncrement and thePageIncrement specify the number of steps to scroll for a line increment and page increment, respectively. isEnabled specifies whether the scrollbar is enabled to receive events. Like all initializers for classes in the CSubview hierarchy, this initializer takes a visibility state and a glue type.

Scrollbar Utilities

```
virtual void SetRange(UNITS theMinPosition,  
UNITS theMaxPosition, UNITS  
theCurrentPosition=SAME);
```

Sets the range for movement of the thumb along the scrollbar. For a vertical scrollbar, theMinPosition is the top; theMaxPosition is the bottom. For a horizontal scrollbar, theMinPosition is the left, and theMaxPosition is the right. theCurrentPosition specifies the current position of the thumb, which must be relative to the minimum and maximum positions.

```
virtual UNITS GetMinPosition(void) const;
```

Sets the minimum position for the scrollbar's thumb. For a horizontal scrollbar, theMinPosition is the left; for a vertical scrollbar, it is the top. The range between is measured in pixels.

```
virtual UNITS GetMaxPosition(void) const;
```

Sets the maximum position for the scrollbar's thumb. For a horizontal scrollbar, theMinPosition is the right; for a vertical scrollbar, it is the bottom. The range between is measured in pixels.

```
virtual void SetPosition(UNITS theNewPosition);
```

Sets the current position of the scrollbar's thumb.

```
virtual UNITS GetPosition(void) const;
```

Returns the position of the scrollbar's thumb.

```
virtual void SetIncrements(UNITS  
theLineIncrements, UNITS  
thePageIncrements);
```

Sets the number of range units that can be scrolled for line increment and page increment events.

virtual UNITS GetLineIncrement(void) const;

Returns the range units that can be scrolled for a line increment event.

virtual UNITS GetPageIncrement(void) const;

Returns the range units that can be scrolled for a page increment event.

virtual void SetOwner(CView* theOwner);

Sets the view that is the owner of the scrollbar. When the scrollbar receives events, it informs its owner by calling the owner's HScroll or VScroll methods.

virtual CView* GetOwner(void) const;

Returns the view that is the owner of the scrollbar. When the scrollbar receives events, it informs its owner by calling the owner's HScroll or VScroll methods. By default, the scrollbar's enclosure acts as its owner.

Static Utility Methods

static UNITS NativeWidth(void);

Returns the optimum native width of a vertical scrollbar. On different platforms, different widths look optimal.

static UNITS NativeHeight(void);

Returns the optimum native height of a horizontal scrollbar.

Inherited Utilities

virtual void DoHit(CONTROL_INFO theControlInfo);

The scrollbar's event-receiving method. It has been overridden to handle any event that comes to the scrollbar and to update the thumb position. When a scrollbar receives a "hit," it takes care of updating its internal state and then propagates the event through a VScroll or HScroll call to its owner. Thus, a scrollbar event is a bidirectional event that is first passed down to the scrollbar and then passed back up to the scrollbar's owner.

DoHit takes a parameter of type CONTROL_INFO, which is defined by XVT in the *XVT Programmer's Guide*. It uses this information to pass the scroll type and the new position to its owner. The possible scroll types are:

SC_NONE	Nowhere. The event is ignored.
SC_LINE_UP	One line up.

SC_LINE_DOWN	One line down.
SC_PAGE_UP	Previous page.
SC_PAGE_DOWN	Next page.
SC_THUMBTRACK	Dynamic thumb tracking.
SC_THUMB	Thumb repositioning.

virtual void Size(const CRect& theNewSize);

An overridden method that resets the region of the scrollbar. Possibly the region could be in a different location and have completely different dimension—a new width or a new height. The coordinates of the region, *theNewSize*, like the coordinates of the region that is passed in on creation of a view, are relative to the enclosure.

Inherited Scrolling Methods

The following two methods are called when a scrollbar receives a scrolling event. The event propagates from the scrollbar to its enclosure. *theEventType* designates the kind of scroll: line up, line down, page up, or page down (the terms “up” and “down” are used for both horizontal and vertical scrolling, though the effects are different for horizontal scrolling). *theThumbPosition* is the numerical value for placement of the thumb when either thumb repositioning or dynamic thumb tracking is used. These methods respond to the scrollbar event by scrolling the views contained inside the scroller.

**virtual void VScroll(SCROLL_CONTROL
theEventType, UNITS thePosition);**

A method called when a scrollbar receives a vertical scroll event.

**virtual void HScroll(SCROLL_CONTROL
theEventType, UNITS thePosition);**

A method called when a scrollbar receives a horizontal scroll event.

Protected Methods

**NScrollBar(WINDOW theCreatedControl, CSubview*
theEnclosure, const CRect& theRegion,
DIRECTION theDirection);**

A constructor that is to be used with derived classes that create a native scrollbar control themselves. *theCreatedControl* is a handle to the scrollbar control that is created by a derived class. *theEnclosure* is a pointer to the subview that will contain the scrollbar. *theRegion* is a coordinate location, local to the

enclosure, that is used to place the scrollbar. theDirection is an enum that takes two possible values: HORIZONTAL or VERTICAL.

virtual SCROLL_TYPE GetSBType(void) const;

Returns the type of the scrollbar. There are three possible types: HSCROLL or VSCROLL, depending on whether the direction is horizontal or vertical, or HVSCROLL, which designates an unattached scrollbar.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Activate(void);
virtual void Deactivate(void);
virtual void DoActivate(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDouble(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,      BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
```



```
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjs(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
```

```

virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);

```

From CNativeView

```

virtual void Disable(void);
virtual void Enable(void);
WIN_TYPE GetXVTType(void) const;
WINDOW GetXVTWindow(void) const;
virtual void Hide(void);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview *thenclosure);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetOrigin(const CPoint& diff);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void UpdateUnits(CUnits* theUnits);

```

Summary: NScrollBar.h

```

#ifndef NScrollBar_H
#define NScrollBar_H

#ifdef STDH
#include "CNativeView.h"
#endif // STDH
#ifdef DOS
#include "CNativVw.h"
#endif // DOS

```

```

class NScrollBar : public CNativeView
{
public:
    // construct, init
    NScrollBar(CSubview* theEnclosure,
        const CRect& theRegion,
        DIRECTION theDirection,
        long theControlAttributes = NULL);

    NScrollBar(const NScrollBar& theScrollBar);
    NScrollBar& operator=(const NScrollBar& theCScrollbar);

    BOOLEAN IScrollBar(UNITS theMinPosition = 0,
        UNITS theMaxPosition = 0,
        UNITS theCurrentPosition = 0,
        UNITS theLineIncrement = 1,
        UNITS thePageIncrement = 20,
        BOOLEAN isEnabled = TRUE,
        BOOLEAN isVisible = TRUE,
        GLUTYPE theGlue = NULLSTICKY);

    // Scrollbar utility:
    virtual void SetRange(UNITS theMinPosition, UNITS theMaxPosition,
        UNITS theCurrentPosition=SAME);

    virtual UNITS GetMinPosition(void) const;
    virtual UNITS GetMaxPosition(void) const;

    virtual void SetPosition(UNITS theNewPosition);
    virtual UNITS GetPosition(void) const;

    virtual void SetIncrements(UNITS theLineIncrements,
        UNITS thePageIncrements);

    virtual UNITS GetLineIncrement(void) const;
    virtual UNITS GetPageIncrement(void) const;

    //Static Utility
    static int NativeWidth(void);
    static int NativeHeight(void);

    // Inherited utility:
    virtual void DoHit(CONTROL_INFO theControlInfo);
    virtual void Size(const CRect& theNewSize);

    virtual void VScroll(SCROLL_CONTROL theEvent, UNITS thePosition);
    virtual void HScroll(SCROLL_CONTROL theEvent, UNITS thePosition);

protected:
    NScrollBar(WINDOW theCreatedControl,
        CSubview* theEnclosure,
        const CRect& theRegion,
        DIRECTION theDirection);

    virtual SCROLL_TYPE GetSBType(void) const;

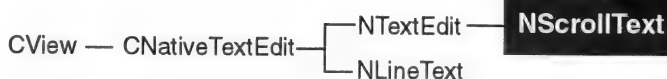
private:

```

```
    UNITS itsPos;           // The scrollbar's position
    UNITS itsMinPosition;   // range start
    UNITS itsMaxPosition;   // range stop
    UNITS itsLineIncrement; // how much to increment on a line scroll
    UNITS itsPageIncrement; // how much to increment on a page scroll
    CView* itsOwner;        // delegate events to owner
};

#endif NScrollBar_H
```

NScrollText



Description

NScrollText adds scrollbars and autoscrolling support to the text editing box.

Heritage

Superclass: NTextEdit

Usage

Create an object of this type and initialize it. See CNativeTextEdit and NTextEdit for more information.

Environment

The text is drawn in the foreground color; the border is drawn with the *pen* color. The interior of the text box is drawn in the background color.

Private Data Members

NScrollBar	*itsHScrollBar;	the horizontal scrollbar
NScrollBar	*itsVScrollBar;	the vertical scrollbar
int	itsViewableLines;	the number of lines that
are		visible within the text
box		
UNITS	itsLineHeight;	
BOOLEAN	itIsThumbtracking;	whether thumbtracking
is		
		supported for the
		scrollbars
BOOLEAN	itUsesWinScrollBars;	whether the scrollbars
are		of type NWinScrollBar

Public Methods

Constructor, Destructor, and Initializer Methods

```
NScrollText(CSubview* theEnclosure,
            const CRect& theRegion,
            BOOLEAN hasHorizontalScrollBar= TRUE,
            BOOLEAN hasVerticalScrollBar = TRUE,
            unsigned theAttributes       = TX_BORDER,
            UNITS theRightMargin         = 1000,
            int theCharacterLimit        = 1000,
            BOOLEAN isVisible             = TRUE);
```

A constructor. It is identical to the corresponding constructor on `NTextEdit`, except that it adds the capability for a horizontal and a vertical scrollbar to the text box (note the two Boolean parameters). This constructor, too, takes a pointer to an enclosure as well as a region that is a `CRect`. The enclosure is the view in which the text box is placed. `theRegion` is a coordinate location that is local to the enclosure, which indicates where the new text box is to be placed. If the text box has a border, which is the default, then the border rectangle is drawn around the `CRect`. This `CRect` object is inset by 4 pixels inside the border. The bottom of the `CRect` may be inset even more to ensure that an integral number of lines will fit into the text box.

In addition, this constructor takes an unsigned value pertaining to the XVT attributes that the text edit system can have. These attributes can be ORed and passed in together, as described in the text editing chapter of the *XVT Programmer's Guide*. Finally, this constructor takes a right margin for the text box, which is measured in pixels starting with zero (0) at the left side of the text box, and a limit on the number of characters that can fit into the text box.

```
NScrollText(CWindow* theScrollableWindow,
            unsigned theAttributes = TX_BORDER,
            UNITS theRightMargin  = 1000,
            int theCharacterLimit  = 1000,
            BOOLEAN isVisible     = TRUE);
```

A constructor. `theScrollableWindow` is a window with scrollbars that will contain the `NScrollText` object. The `NScrollText` object will occupy the entire window's client area and will be operated using the window's scrollbars. See the description of the previous constructor for an explanation of the remaining parameters.

```
NScrollText(const NScrollText& theScrollText);
```

A copy constructor that duplicates the attributes of a text box, including its scrollbars, as well as the text inside it.

```
NScrollText& operator=(const NScrollText& theScrollText);
```

An assignment operator that duplicates the attributes of a text box, including its scrollbars, as well as the text inside it.

```
virtual ~NScrollText(void);
```

The destructor, which cleans up the text edit system.

```
BOOLEAN IScrollText(BOOLEAN isThumbTracking      = TRUE,  
                     unsigned theAttributes        = TX_BORDER,  
                     UNITS theRightMargin          = 1000,  
                     int theCharacterLimit         = 1000,  
                     const CString theInitialText = NULLString,  
                     BOOLEAN isAutoSelected       = FALSE,  
                     BOOLEAN isVisible            = TRUE,  
                     long theGlue                  = NULLSTICKY);
```

The initializer. It is identical to the initializer of `NEditText`, except that it adds the capacity to support thumbtracking for the scrollbars (note the Boolean parameter). It takes an unsigned value pertaining to the XVT attributes that the text edit system can have. These attributes can be ORed and passed in together, as described in the text editing chapter of the *XVT*

Programmer's Guide. For further information on the possible values for theAttributes, see the table in the section on `CNativeTextEdit`, under `INativeTextEdit`.

The next parameter, `theRightMargin` is a pixel measurement of the right margin, starting from the left side and beginning at zero (0). In addition, there is a parameter that sets the default character limit at 1,000 characters. `theInitialText` is a string containing any text you may wish to initialize with the text edit system. If the `isAutoSelected` parameter is set to `TRUE`, all of the text in the text edit box becomes selected when the user clicks inside the box. Finally, this initializer, like the initializers of all `CView` classes, takes a visibility state and a glue type.

Scrolling Functions

```
virtual void SetHIncrements(UNITS  
                     theLineIncrement, UNITS thePageIncrement);
```

Sets the horizontal increments of the scroller, in text lines, for both line increments and page increments.

```
virtual void SetVIncrements(UNITS  
    theLineIncrement, UNITS thePageIncrement);
```

Sets the vertical increments of the scroller, in text lines, for both line increments and page increments.

```
virtual UNITS GetHLineIncrement(void) const;
```

Returns the number of lines that have been set for a horizontal line increment during scrolling.

```
virtual UNITS GetVLineIncrement(void) const;
```

Returns the number of lines that have been set for a vertical line increment during scrolling.

```
virtual UNITS GetHPageIncrement(void) const;
```

Returns the number of lines that have been set for a horizontal page increment during scrolling.

```
virtual UNITS GetVPageIncrement(void) const;
```

Returns the number of pixels that have been set for a vertical page increment during scrolling.

Scrolling Event Methods

The following two methods are called when a scrollbar receives a scrolling event. The event propagates from the scrollbar to the CScroller object. `theEventType` designates the kind of scroll: line up, line down, page up, or page down (the terms “up” and “down” are used for both horizontal and vertical scrolling, though the effects are different for horizontal scrolling). `thePosition` is the numerical value for placement of the thumb when either thumb repositioning or dynamic thumb tracking is used. These methods respond to the scrollbar event by scrolling the views contained inside the scroller.

```
virtual void VScroll(SCROLL_CONTROL  
    theEventType, UNITS thePosition);
```

A method called when a scrollbar receives a vertical scroll event.

```
virtual void HScroll(SCROLL_CONTROL  
    theEventType, UNITS thePosition);
```

A method called when a scrollbar receives a horizontal scroll event.

Inherited Utilities

```
virtual void SetMargin(UNITS theRightMargin);
```

Sets the right margin of a text edit box. `theRightMargin` is a pixel measurement of the right margin, starting from the left

side and beginning at zero (0). If the NScrollText object has a horizontal scrollbar, this method resets the scrollbar's range according to the new margin.

```
virtual void SetFont(const FONT& theNewFont,  
    BOOLEAN isUpdate = FALSE);
```

Takes a font and sets the text box's environment to have that font. The isUpdate parameter indicates whether this SetFont message is simply an update message or whether it is a "real" SetFont message meaning that the text box should set its own font to the new font.

```
virtual void Size(const CRect& theNewSize);
```

Resets the region of the text box. Possibly the region could be in a different location and have completely different dimensions—a new width or a new height. The coordinates of the region, theNewSize, like the coordinates of the region that is passed in on creation of a text box, are relative to the enclosure.

Protected Methods

None.

Private Methods

```
void CreateScrollBars(BOOLEAN hasVScrollBar,  
    BOOLEAN hasHScrollBar, UNITS  
    theLineIncrement, UNITS thePageIncrement);
```

An internal method that is called upon construction of a text box with scrollbars. The first two parameters take a Boolean value that sets whether the text box has a vertical and/or horizontal scrollbars. theLineIncrement specifies the number of pixels to scroll for a line increment; similarly, thePageIncrement specifies the number of pixels to scroll for a page increment.

```
static void ScrollTextCallback(TXEDIT theTextEdit, T_LNUM  
    theOriginLine, T_LNUM theNumberOfLines, T_CNUM  
    theCharacterOffset);
```

A method that is called by XVT when the contents of the text box have been scrolled. It updates the scrollbars. theTextEdit identifies the specific text object that needs updating. This method also takes the number of the starting line (numbering starts at zero from the top line in the text edit object and extends across paragraphs). theNumberOfLines is the total number of lines contained within the text edit object. Finally, theCharacterOffset designates the number of pixels to the left of the view rectangle.

```
CRect GetInitialFrame(CSubview* theEnclosure,  
const CRect& theRegion, BOOLEAN  
hasHorizontalScrollBar, BOOLEAN  
hasVerticalScrollBar);
```

An internal method that calculates the initial size of the object during construction, also noting whether it has a horizontal and/or vertical scrollbar.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);  
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);  
virtual CUnits* GetUnits(void) const;  
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Deactivate(void);  
virtual void DoActivate(void);  
virtual void DoCommand(long theCommand, void* theData=NULL);  
virtual void DoDeactivate(void);  
virtual void DoDisable(void);  
virtual void DoDraw(void);  
virtual void DoDraw(const CRect& theClippingRegion);  
virtual void DoEnable(void);  
virtual void DoHide(void);  
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);  
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN  
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);  
virtual void DoSetDragging(BOOLEAN isDraggable);  
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);  
virtual void DoSetGlue(GLUETYPE theGlue);
```

```
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
```

```

virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);

void Prepare(const CRect& theClippingRegion);

virtual void SetCommand(long theCommand);

virtual void SetDoubleCommand(long theCommand);

virtual void SetDragging(BOOLEAN isDraggable);

virtual void SetEnclosure(CSubview* theEnclosure);

virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);

virtual void SetId(int theID);

virtual void SetSizing(BOOLEAN isSizable);

virtual void SetTitle(const CString& theNewTitle);

virtual void SetWireFrame(CWireFrame* theNewWireFrame);

```

From CNativeTextEdit

```

virtual void Activate(void);

virtual BOOLEAN Append(const CString& theText);

virtual BOOLEAN Clear(void);

virtual void Disable(void);

virtual void Draw(const CRect& theClipping);

virtual unsigned long GetAttributes(void) const;

void GetFullPar(T_PNUM theParagraph, CString& theText) const;

void GetLineInternal(T_PNUM theParagraph, T_LNUM theLine,    CString& theText) const;

virtual int GetMargin(void) const;

virtual T_CNUM GetNCharInSelection(void) const;

T_CNUM GetNCharInternal(T_PNUM theFirstParagraph, T_PNUM theLastParagraph, T_LNUM
theFirstLine, T_LNUM theLastLine, T_CNUM theFirstChar, T_CNUM theLastChar) const;

virtual T_CNUM GetNCharInText(void) const;

virtual int GetLimit(void) const;

void GetPartLine(T_CNUM theFirstChar,T_CNUM theLastChar, T_PNUM
theParagraph,T_LNUM theLineNumber, CString& theText) const;

void GetPartPar(T_CNUM theStartChar,T_CNUM theEndChar, T_PNUM theParagraph, T_LNUM
theStartLine,T_LNUM theEndLine,CString& theText) const;

virtual CString GetSelectedText(void) const;

virtual CString GetText(void) const;

void GetTextInternal(T_PNUM theFirstParagraph, T_PNUM theLastParagraph, T_LNUM
theFirstLine, T_LNUM theLastLine, T_CNUM theFirstChar, T_CNUM theLastChar,
CString& theTextBuffer) const;

```

```

virtual void Hide(void);
BOOLEAN IsEmpty(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton=0, BOOLEAN isShiftKey = FALSE, BOOLEAN isControlKey = FALSE);
virtual void MouseDown(CPoint theLocation, short theButton=0,    BOOLEAN isShiftKey = FALSE, BOOLEAN isControlKey = FALSE);
virtual void SelectText(void);
virtual void SetAttribute(unsigned theAttribute,BOOLEAN isSet=TRUE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetLimit(int theCharacterLimit);
virtual void SetOrigin(const CPoint& theDelta);
virtual void SetText(const CString& theText);
virtual void Show(void);
void Truncate(CString& theText);
virtual void UpdateUnits(CUnits* theUnits);
virtual int Validate(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);

```

From NTextEdit

```

virtual BOOLEAN AddParagraph(const CString& theText, T_PNUM theParagraph);
virtual BOOLEAN AppendToParagraph(const CString& theText,    T_PNUM theParagraph);
virtual CString GetLine(T_PNUM theParagraph, T_LNUM theLine) const;
virtual T_CNUM GetNCharInline(T_PNUM theParagraph=0, T_LNUM theLine=0) const;
virtual T_CNUM GetNCharInPar(T_PNUM theParagraph) const;
virtual T_LNUM GetNLineInPar(T_PNUM theParagraph) const;
virtual T_LNUM GetNLineInSelection(void) const;
virtual T_LNUM GetNLineInText(void) const;
virtual T_PNUM GetNParInSelection(void) const;
virtual T_PNUM GetNParInText(void) const;
virtual CString GetParagraph(T_PNUM theParagraph) const;
virtual CString GetSomeText(T_PNUM theStartParagraph,T_PNUM theEndParagraph,T_LNUM theStartLine,T_LNUM theEndLine,T_CNUM theStartChar, T_CNUM theEndChar) const;
virtual void SelectLine(T_LNUM theLine=0, T_PNUM theParagraph=0, T_CNUM theStartingChar=0, T_CNUM theEndingChar=LAST);
virtual void SelectSomeText(T_PNUM theStartParagraph, T_PNUM theEndParagraph,T_LNUM theStartLine,T_LNUM theEndLine, T_CNUM theStartChar, T_CNUM theEndChar);
virtual void SetLine(const CString& theText, T_PNUM theParagraph, T_LNUM theLine);
virtual BOOLEAN SetParagraph(const CString& theText,T_PNUM theParagraph);

```

Summary: NScrollText.h

```

#ifndef NScrollText_H
#define NScrollText_H

#ifdef STDH
#include "NTextEdit.h"
#endif STDH
#ifdef DOS
#include "NTextEdt.h"
#endif DOS

class CTextScrollBar;

class NScrollText : public NTextEdit
{
public:
    // construct & destruct & init
    NScrollText(CSubview* theEnclosure, const CRect& theRegion,
        BOOLEAN hasHorizontalScrollBar = TRUE,
        BOOLEAN hasVerticalScrollBar = TRUE,
        unsigned theAttributes = TX_BORDER,
        UNITS theRightMargin = 1000,
        int theCharacterLimit = 1000,
        BOOLEAN isVisible = TRUE);

    NScrollText(CWindow* theScrollableWindow,
        unsigned theAttributes = TX_BORDER,
        UNITS theRightMargin = 1000,
        int theCharacterLimit = 1000,
        BOOLEAN isVisible = TRUE);

    NScrollText(const NScrollText& theScrollText);
    NScrollText& operator=(const NScrollText& theScrollText);
    virtual ~NScrollText(void);

    BOOLEAN IScrollText(BOOLEAN isThumbTracking = TRUE,
        unsigned theAttributes = TX_BORDER,
        UNITS theRightMargin = 1000,
        int theCharacterLimit = 1000,
        const CString theInitialText = NULLString,
        BOOLEAN isAutoSelected = FALSE,
        BOOLEAN isVisible = TRUE,
        long theGlue = NULLSTICKY);

    // Scrolling Functions:
    virtual void SetHIncrements(UNITS theLineIncrement,
        UNITS thePageIncrement);
    virtual void SetVIncrements(UNITS theLineIncrement,
        UNITS thePageIncrement);

    virtual UNITS GetHLineIncrement(void) const;
    virtual UNITS GetVLineIncrement(void) const;
    virtual UNITS GetHPageIncrement(void) const;
    virtual UNITS GetVPageIncrement(void) const;

    // Scroll Event Methods:
    virtual void VScroll(SCROLL_CONTROL theEventType,
        UNITS thePosition);
    virtual void HScroll(SCROLL_CONTROL theEventType,
        UNITS thePosition);

```

```
// Inherited Utilities:
virtual void SetMargin(UNITS theRightMargin);
virtual void SetFont(const FONT& theNewFont,
    BOOLEAN isUpdate = FALSE);
virtual void Size(const CRect& theNewSize);

private:
    NScrollBar *itsHScrollBar;
    NScrollBar *itsVScrollBar;
    int itsViewableLines;
    UNITS itsLineHeight;
    BOOLEAN itIsThumbtracking;
    BOOLEAN itUsesWinScrollBars;

    void CreateScrollBars(BOOLEAN hasVScrollBar,
        BOOLEAN hasHScrollBar,
        UNITS theLine, UNITS thePage);

    static void ScrollTextCallback(TXEDIT theTextEdit,
        T_LNUM theOriginLine,
        T_LNUM theNumberOfLines,
        T_CNUM theCharacterOffset);

    CRect GetInitialFrame(CSubview* theEnclosure,
        const CRect& theRegion,
        BOOLEAN hasHorizontalScrollBar,<class>
        BOOLEAN hasVerticalScrollBar);
};

#endif // NScrollText_H
```

NTextEdit



Description

NTextEdit adds the capability to organize the text inside a text box into paragraphs and lines. NTextEdit objects consist of multi-line text editing boxes that can be placed inside CViews. They provide full editing support, including cut, copy, and paste operations.

Heritage

Superclass: CNativeTextEdit

Subclass: NScrollText

Usage

Create and initialize an NTextEdit object. Users can then type inside the editing box.

Environment

The text is drawn in the foreground color; the border is drawn with the *pen* color. The interior of the text box is drawn in the background color.

Data Members

None.

Public Methods

Constructor, Destructor, and Initializer Methods

```

NTextEdit(CSubview* theEnclosure, const CRect& theRegion,
          unsigned theAttributes           = TX_BORDER,
          int theRightMargin              = 1000,
          int theCharacterLimit            = 1000);
  
```

A constructor. theEnclosure is a pointer to the subview that will contain the text box. theRegion is a coordinate location, local to the enclosure, that is used to place the text box. If the text box has a border, which is the default, then the border rectangle is drawn around the CRect. This CRect object is inset

by 4 pixels inside the border. The bottom of the CRect may be inset even more to ensure that an integral number of lines will fit into the text box.

In addition, this constructor takes an unsigned value pertaining to the XVT attributes that the text edit system can have. These attributes can be ORed and passed in together, as described in the text editing chapter of the *XVT Programmer's Guide*. For further information on the possible values for theAttributes, see the table in the section on CNativeTextEdit, under INativeTextEdit.

Finally, this constructor takes a right margin for the text box, which is measured in pixels starting with zero (0) at the left side of the text box, and a limit on the number of characters that can fit into the text box.

NTextEdit(const NTextEdit& theTextEdit);

A copy constructor that duplicates the attributes of a text box, as well as the text inside it.

NTextEdit& operator=(const NTextEdit& theTextEdit);

An assignment operator that duplicates the attributes of a text box, as well as the text inside it.

virtual ~NTextEdit(void);

The destructor, which cleans up the text edit system.

```

BOOLEAN ITextEdit(unsigned theAttributes      = TX_BORDER,
int theRightMargin      = 1000,
int theCharacterLimit   = 1000,
const CString theInitialText= NULLString,
BOOLEAN isAutoSelected  = FALSE,
BOOLEAN isVisible      = TRUE,
long theGlue            = NULLSTICKY);

```

The initializer. It is identical to the initializer of CNativeTextEdit. It takes an unsigned value pertaining to the XVT attributes that the text edit system can have. These attributes can be ORed and passed in together, as described in the text editing chapter of the *XVT Programmer's Guide*. For further information on the possible values for theAttributes, see the table under INativeTextEdit.

The next parameter, theRightMargin is a pixel measurement of the right margin, starting from the left side and beginning at zero (0). In addition, there is a parameter that sets the default character limit at 1,000 characters. theInitialText is a string containing any text you may wish to initialize with the text edit

system. If the `isAutoSelected` parameter is set to `TRUE`, all of the text in the text edit box becomes selected when the user clicks inside the box. Finally, this initializer, like the initializers of all `CView` classes, takes a visibility state and a glue type.

Paragraph Methods

```
virtual BOOLEAN SetParagraph(const CString&  
theText, T_PNUM theParagraph);
```

Sets a paragraph. This method takes a string buffer and a paragraph number. The paragraph number must be a valid, existing paragraph number.

```
virtual BOOLEAN AddParagraph(const CString&  
theText, T_PNUM theParagraph);
```

Adds a paragraph to a text box. It takes a string buffer and a new paragraph number. If `(theParagraph==USHRT_MAX)`, the paragraph is added at the end of the text.

```
virtual BOOLEAN AppendToParagraph(const  
CString& theText, T_PNUM theParagraph);
```

Appends a paragraph to a text box. It takes a string buffer and a paragraph number. The paragraph number must be a valid, existing paragraph number.

```
virtual BOOLEAN DeleteParagraph(T_PNUM  
theParagraph);
```

Removes the paragraph designated by `theParagraph` from the text box. Numbering of paragraphs starts at zero. This method returns `TRUE` if the paragraph is successfully deleted.

```
virtual CString GetParagraph(T_PNUM  
theParagraph) const;
```

Given a paragraph number, `GetParagraph` returns the designated paragraph.

```
virtual void SelectParagraph(T_PNUM theParagraph,  
                             T_LNUM theStartLine = 0,  
                             T_LNUM theEndLine   = LAST,  
                             T_CNUM theStartChar = 0,  
                             T_CNUM theEndChar   = LAST);
```

Selects a paragraph within a text box. This method takes a paragraph number, the numbers of the first and last lines in the paragraph, and the numbers of the starting and ending characters in the paragraph.

```
virtual T_CNUM GetNCharInPar(T_PNUM  
    theParagraph) const;
```

Given a paragraph number, returns the total number of characters in that paragraph.

```
virtual T_PNUM GetNParInText(void) const;
```

Returns the total number of paragraphs inside a text box.

```
virtual T_PNUM GetNParInSelection(void) const;
```

Returns the total number of paragraphs in the selected text.

Line Methods

```
virtual void SetLine(const CString& theText,  
    T_PNUM theParagraph, T_LNUM theLine);
```

Sets a line within a paragraph. This method takes a string buffer, a paragraph number, and a line number. Both the paragraph number and the line number must be valid, existing numbers.

```
virtual CString GetLine(T_PNUM theParagraph,  
    T_LNUM theLine) const;
```

Given a paragraph number and a line number, GetLine returns the designated line.

```
virtual void SelectLine(T_LNUM theLine=0,  
    T_PNUM theParagraph=0, T_CNUM  
    theStartingChar=0, T_CNUM  
    theEndingChar=LAST);
```

Selects a line of text. This method takes a line number (starting at zero (0) from the top of the text box), a paragraph number, and numbers for the starting and ending characters.

```
virtual T_CNUM GetNCharInLine(T_PNUM  
    theParagraph=0, T_LNUM theLine=0) const;
```

Given a paragraph number and a line number, this method returns the total number of characters in the line.

```
virtual T_LNUM GetNLineInPar(T_PNUM  
    theParagraph) const;
```

Given a paragraph number, this method returns the total number of lines in that paragraph.

```
virtual T_LNUM GetNLineInText(void) const;
```

Returns the total number of lines contained in a text box.

```
virtual T_LNUM GetNLineInSelection(void) const;
```

Returns the total number of lines contained in the selected text.

Partial Text Utility

```
virtual CString GetSomeText(T_PNUM theStartParagraph,
                           T_PNUM theEndParagraph,
                           T_LNUM theStartLine,
                           T_LNUM theEndLine,
                           T_CNUM theStartChar,
                           T_CNUM theEndChar);
```

Returns a requested body of text. GetSomeText takes the numbers of the starting and ending paragraphs, the numbers of the starting and ending lines, and the numbers of the starting and ending characters.

```
virtual void SelectSomeText(T_PNUM theStartParagraph,
                            T_PNUM theEndParagraph,
                            T_LNUM theStartLine,
                            T_LNUM theEndLine,
                            T_CNUM theStartChar,
                            T_CNUM theEndChar);
```

Selects a specified body of text. This method takes the numbers of the starting and ending paragraphs, the numbers of the starting and ending lines, and the numbers of the starting and ending characters.

Methods Inherited But Not Overridden

Methods are listed alphabetically by name under their respective classes.

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN
isControlKey);
virtual CUnits* GetUnits(void) const;
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Deactivate(void);
virtual void DoActivate(void);
virtual void DoCommand(long theCommand, void* theData=NULL);
virtual void DoDeactivate(void);
virtual void DoDisable(void);
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
```

```
virtual void DoHide(void);
virtual void DoKey(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDoubleClick(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Enable(void);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual long GetDoubleCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
```

```

virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString& GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual void HScroll(SCROLL_CONTROL theType, int thePos);
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible(void) const;
virtual void MouseMove(CPoint theLocation, short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation, short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetEnclosure(CSubview* theEnclosure);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetId(int theID);
virtual void SetSizing(BOOLEAN isSizable);
virtual void SetTitle(const CString& theNewTitle);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);
virtual void VScroll(SCROLL_CONTROL theType, int thePos);

```

From CNativeTextEdit

```

virtual void Activate(void);
virtual BOOLEAN Append(const CString& theText);
virtual BOOLEAN Clear(void);
virtual void Disable(void);
virtual void Draw(const CRect& theClipping);
virtual unsigned long GetAttributes(void) const;
void GetFullPar(T_PNUM theParagraph, CString& theText) const;

```

```

void GetLineInternal(T_PNUM theParagraph, T_LNUM theLine, CString& theText) const;
virtual int GetMargin(void) const;
virtual T_CNUM GetNCharInSelection(void) const;
T_CNUM GetNCharInternal(T_PNUM theFirstParagraph, T_PNUM theLastParagraph, T_LNUM
theFirstLine, T_LNUM theLastLine, T_CNUM theFirstChar, T_CNUM theLastChar) const;
virtual T_CNUM GetNCharInText(void) const;
virtual int GetLimit(void) const;
void GetPartLine(T_CNUM theFirstChar, T_CNUM theLastChar, T_PNUM
theParagraph, T_LNUM theLineNumber, CString& theText) const;
void GetPartPar(T_CNUM theStartChar, T_CNUM theEndChar, T_PNUM theParagraph, T_LNUM
theStartLine, T_LNUM theEndLine, CString& theText) const;
virtual CString GetSelectedText(void) const;
virtual CString GetText(void) const;
void GetTextInternal(T_PNUM theFirstParagraph, T_PNUM theLastParagraph, T_LNUM
theFirstLine, T_LNUM theLastLine, T_CNUM theFirstChar, T_CNUM theLastChar,
CString& theTextBuffer) const;
virtual void Hide(void);
BOOLEAN IsEmpty(void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation, short theButton=0, BOOLEAN isShiftKey
= FALSE, BOOLEAN isControlKey = FALSE);
virtual void MouseDown(CPoint theLocation, short theButton=0, BOOLEAN isShiftKey =
FALSE, BOOLEAN isControlKey = FALSE);
virtual void SelectText(void);
virtual void SetAttribute(unsigned theAttribute, BOOLEAN isSet=TRUE);
virtual void SetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetLimit(int theCharacterLimit);
virtual void SetMargin(int theRightMargin);
virtual void SetOrigin(const CPoint& theDelta);
virtual void SetText(const CString& theText);
virtual void Show(void);
virtual void Size(const CRect& theNewSize);
void Truncate(CString& theText);
virtual void UpdateUnits(CUnits* theUnits);
virtual int Validate(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);

```

Summary: NTextEdit.h

```

#ifndef NTextEdit_H
#define NTextEdit_H

#ifdef STDH
#include "CNativeTextEdit.h"
#endif STDH
#ifdef DOS
#include "CNTvTxE.h"
#endif DOS

class NTextEdit : public CNativeTextEdit
{
public:
    // construc & destruct & init
    NTextEdit(CSubview* theEnclosure, const CRect& theRegion,
              unsigned theAttributes      = TX_BORDER,
              UNITS theRightMargin        = 1000,
              int theCharacterLimit        = 1000);
    NTextEdit(const NTextEdit& theTextEdit);
    NTextEdit& operator=(const NTextEdit& theTextEdit);

    BOOLEAN ITextEdit(unsigned theAttributes = TX_BORDER,
                      UNITS theRightMargin   = 1000,
                      int theCharacterLimit   = 1000,
                      const CString& theInitialText = NULLString,
                      BOOLEAN isAutoSelected   = FALSE,
                      BOOLEAN isVisible        = TRUE,
                      GLUETYPE theGlue        = NULLSTICKY);

    // Paragraphs:
    virtual BOOLEAN SetParagraph(const CString& theText,
                                T_PNUM theParagraph);
    virtual BOOLEAN AddParagraph(const CString& theText,
                                T_PNUM theParagraph);
    virtual BOOLEAN AppendToParagraph(const CString& theText,
                                      T_PNUM theParagraph);
    virtual BOOLEAN DeleteParagraph(T_PNUM theParagraph);
    virtual CString GetParagraph(T_PNUM theParagraph) const;
    virtual void SelectParagraph(T_PNUM theParagraph,
                                T_LNUM theStartLine=0,
                                T_LNUM theEndLine=LAST,
                                T_CNUM theStartChar=0,
                                T_CNUM theEndChar=LAST);

    virtual T_CNUM GetNCharInPar(T_PNUM theParagraph) const;
    virtual T_PNUM GetNParInText(void) const;
    virtual T_PNUM GetNParInSelection(void) const;

    // Lines:
    virtual void SetLine(const CString& theText,
                        T_PNUM theParagraph, T_LNUM theLine);
    virtual CString GetLine(T_PNUM theParagraph, T_LNUM theLine) const;

```



```
virtual void Selectline(T_LNUM theLine=0, T_PNUM theParagraph=0,
                      T_CNUM theStartingChar=0,
                      T_CNUM theEndingChar=LAST);

virtual T_CNUM GetNCharInLine(T_PNUM theParagraph=0,
                             T_LNUM theLine=0) const;
virtual T_LNUM GetNLineInPar(T_PNUM theParagraph) const;
virtual T_LNUM GetNLineInText(void) const;
virtual T_LNUM GetNLineInSelection(void) const;

// Partial text utility:

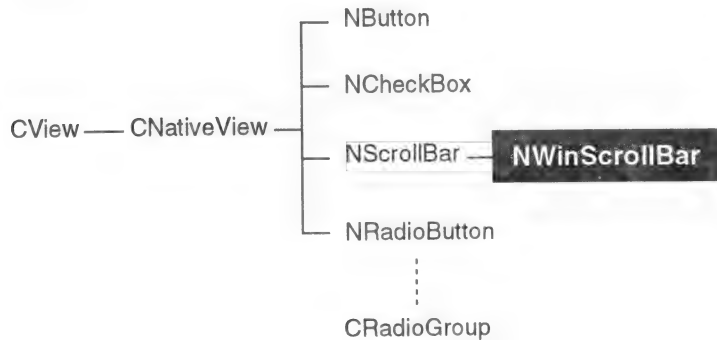
virtual CString GetSomeText(T_PNUM theStartParagraph,
                           T_PNUM theEndParagraph,
                           T_LNUM theStartLine, T_LNUM theEndLine,
                           T_CNUM theStartChar, T_CNUM theEndChar)
    const;

virtual void SelectSomeText(T_PNUM theStartParagraph,
                           T_PNUM theEndParagraph,
                           T_LNUM theStartLine, T_LNUM theEndLine,
                           T_CNUM theStartChar, T_CNUM theEndChar);

};

#endif // NTextEdit_H
```

NWinScrollBar



Description

NWinScrollBar is a class whose objects are instantiated internally whenever windows with scrollbars are created. Windows with scrollbars are special types of windows in that the scrollbars are part of the window itself. In contrast, scrollbars of the NScrollBar class are not attached to the window and can be resized or moved around inside the window. Classes such as CScroller and NScrollText sometimes use NWinScrollBar whenever they are created with a special constructor that informs the scroller or scrolltext object that it should use the window's attached scrollbars instead of their own unattached scrollbars.

Heritage

Superclass: NScrollBar

Usage

You cannot create an NWinScrollBar directly. Instead, you create a window and specify as one of its XVT attributes WSF_VSCROLL or WSF_HSCROLL. The window will have scrollbars, and objects of the NWinScrollBar class will be instantiated to manage those scrollbars.

Environment

There are no environment settings for classes that inherit from CNativeView. The drawing properties of native views are system-defined.

Data Members

None.

Public Methods

Constructor Methods

```
NWinScrollBar(CSubview*theEnclosure,DIRECTION  
theDirection);
```

A constructor. `theEnclosure` is a pointer to the view that is the enclosure of the scrollbar, usually a window. `theDirection` indicates whether the scrollbar is horizontal or vertical.

```
NWinScrollBar(const NWinScrollBar&  
theScrollBar);
```

A copy constructor. It disables copying of the scrollbar. You cannot copy one of these scrollbars to another because you cannot instantiate them directly. They are part of a window, and the window creates them.

```
NWinScrollBar& operator=(const NWinScrollBar&  
theCScrollBar);
```

An assignment operator, which, like the copy constructor, disables copying of the scrollbar.

Inherited and Disabled Methods

```
virtual void Size(const CRect& theNewSize);
```

You cannot size a window's scrollbar. It is sized automatically when the window is sized.

```
virtual void SetTitle(const CString& theTitle);
```

You cannot set the title of scrollbars.

```
virtual void Hide(void);
```

You cannot hide a scrollbar. It is automatically hidden when the window to which it is attached is hidden. On some platforms, such as Microsoft Windows, the scrollbars may appear or disappear as standard behavior dictates, depending on whether scrolling is needed.

```
virtual void Show(void);
```

You cannot show a scrollbar. It is automatically revealed when the window to which it is attached is revealed.

```
virtual void SetId(int theId);
```

You cannot set the ID of an `NWinScrollBar` object.

```
virtual void SetEnclosure(CSubview  
*theEnclosure);
```

You cannot set the enclosure of an NWinScrollBar object.

```
virtual void Enable(void);
```

You cannot enable an NWinScrollBar object.

```
virtual void Disable(void);
```

You cannot disable an NWinScrollBar object.

Protected Methods

```
virtual SCROLL_TYPE GetSBType(void) const;
```

Returns the type of the scrollbar. There are three possible types: HSCROLL or VSCROLL, depending on whether the direction is horizontal or vertical, or HVSCROLL, which designates an unattached scrollbar.

Private Methods

Methods Inherited From CNativeView

Both of the following inherited methods are disabled.

```
virtual void Close(void);
```

You cannot close a scrollbar.

```
virtual void CreateControl(void);
```

You cannot re-create a scrollbar.

Methods Inherited But Not Overridden

From CBoss

```
virtual void ChangeFont(FONT theFont, FONT_PART theChange);
```

```
virtual void DoMenuCommand(MENU_TAG theMenuItem, BOOLEAN isShiftKey, BOOLEAN  
isControlKey);
```

```
virtual CUnits* GetUnits(void) const;
```

```
virtual void SetUnits (CUnits* theCoordinateUnits);
```

From CView

```
virtual void Activate(void);
```

```
virtual void Deactivate(void);
```

```
virtual void DoActivate(void);
```

```
virtual void DoCommand(long theCommand, void* theData=NULL);
```

```
virtual void DoDeactivate(void);
```

```
virtual void DoDisable(void);
```

```
virtual void DoDraw(void);
virtual void DoDraw(const CRect& theClippingRegion);
virtual void DoEnable(void);
virtual void DoHide(void);
virtual void DoKey(int theKey,BOOLEAN isShiftKey,BOOLEAN isControlKey);
virtual void DoMouseDown(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseMove(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoMouseUp(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void DoSetDragging(BOOLEAN isDraggable);
virtual void DoSetFont(const FONT& theNewFont, BOOLEAN isUpdate = FALSE);
virtual void DoSetGlue(GLUETYPE theGlue);
virtual void DoSetOrigin(const CPoint& theDeltaPoint);
virtual void DoSetSizing(BOOLEAN isSizable);
virtual void DoShow(void);
virtual void DoSize(const CRect& theNewSize);
virtual void DoTimer(long theTimerId);
virtual void DoUser(long theUserId, void* theData);
virtual void Draw(void);
virtual void Draw(const CRect& theClippingRegion);
virtual CView* FindDeepSubview(const CPoint& theLocation) const;
virtual CView* FindHitView(const CPoint& theLocation) const;
virtual CView* FindSubview(const CPoint& theLocation) const;
virtual CRect GetClippedFrame(void) const;
virtual long GetCommand(void) const;
virtual CWindow* GetCWindow(void) const;
virtual long GetDoubleCommand(void) const;
virtual CSubview* GetEnclosure(void);
virtual const CEnvironment* GetEnvironment(void) const;
virtual CRect GetFrame(void) const;
virtual CRect GetGlobalFrame(void) const;
virtual CPoint GetGlobalOrigin(void) const;
virtual GLUETYPE GetGlue(void) const;
```

```

virtual int GetId(void) const;
virtual CRect GetLocalFrame(void) const;
virtual CPoint GetOrigin(void) const;
virtual CView* GetSelectedView(void) const;
virtual const CList* GetSubObjects(void) const = NULL;
virtual const CString GetTitle(void) const;
virtual CWireFrame* GetWireFrame(void) const;
virtual void Glue();
virtual BOOLEAN IsActive(void) const;
virtual BOOLEAN IsEnabled(void) const;
virtual BOOLEAN IsEnvironmentShared(void);
virtual BOOLEAN IsDraggable(void) const;
virtual BOOLEAN IsSizable(void) const;
virtual BOOLEAN IsVisible (void) const;
virtual void Key(int theKey, BOOLEAN isShiftKey, BOOLEAN isControlKey);
virtual void MouseDouble(CPoint theLocation,short theButton = 0,BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseDown(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseMove(CPoint theLocation,short theButton = 0,    BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
virtual void MouseUp(CPoint theLocation,short theButton = 0, BOOLEAN
isShiftKey=FALSE, BOOLEAN isControlKey=FALSE);
void Prepare(const CRect& theClippingRegion);
virtual void SetCommand(long theCommand);
virtual void SetDoubleCommand(long theCommand);
virtual void SetEnvironment(const CEnvironment& theNewEnvironment, BOOLEAN
isUpdate = FALSE);
virtual void SetFont(const FONT& font, BOOLEAN isUpdate = FALSE);
virtual void SetWireFrame(CWireFrame* theNewWireFrame);

```

From CNativeView

```

WIN_TYPE GetXVTType(void) const;
WINDOW GetXVTWindow(void) const;
virtual void SetDragging(BOOLEAN isDraggable);
virtual void SetGlue(GLUETYPE theGlue);
virtual void SetOrigin(const CPoint& diff);
virtual void SetSizing(BOOLEAN isSizable);

```

```
virtual void UpdateUnits(CUnits* theUnits);
```

From NScrollBar

```
virtual void DoHit(CONTROL_INFO theControlInfo);
```

```
virtual void HScroll(SCROLL_CONTROL theEvent, int thePosition);
```

```
virtual void VScroll(SCROLL_CONTROL theType, int thePos);
```

Summary: NWinScrollBar.h

```
#ifndef NWinScrollBar_H
#define NWinScrollBar_H

#include "PwrDef.h"
#include NScrollBar_i

class NWinScrollBar : public NScrollBar
{
public:
    // construct, init
    NWinScrollBar(CSubview* theEnclosure, DIRECTION theDirection);
    NWinScrollBar(const NWinScrollBar& theScrollBar);
    NWinScrollBar& operator=(const NWinScrollBar& theCScrollBar);

    // Inherited and Disabled
    virtual void Size(const Rect& theNewSize);
    virtual void SetTitle(const CString& theTitle);
    virtual void Hide(void);
    virtual void Show(void);
    virtual void SetId(int theId);
    virtual void SetEnclosure(CSubview *theEnclosure);
    virtual void Enable(void);
    virtual void Disable(void);

protected:
    virtual SCROLL_TYPE GetSBType(void) const;

private:
    virtual void Close(void);
    virtual void CreateControl(void);
};

#endif // NWinScrollBar_H
```


XVT-POWER++

INDEX

A

abstract classes

- CApplication, 8
- CBoss, 29
- CDialog, 54
- CDocument, 64
- CGrid, 109
- CNativeList, 181
- CNativeSelectList, 188
- CNativeTextEdit, 195
- CNativeView, 210
- CShape, 304
- CView, 421
- CVirtualFrame, 441

accessing data, 64

Activate

- CNativeTextEdit, 201
- CView, 423
- NListEdit, 509

activating the File menu, 12

AddButton, 514

- CRadioGroup, 253

AddButtonInternal

- CRadioGroup, 256

AddButtons, 514

- CRadioGroup, 254

AddDocument, 17

adding button behavior to an icon, 35

AddParagraph

- NTextEdit, 544

AddSubview

- CScroller, 290
- CSubview, 366
- CVirtualFrame, 444

AddWindow

- CDesktop, 51
- CDocument, 73

AdjustCells

- CGrid, 116

AdjustCol

- CVariableGrid, 413

adjusting grids, 110

AdjustInternalPoints

- CPolygon, 243

AdjustRow

- CVariableGrid, 413

allocation of memory, 171

Append

- CNativeTextEdit, 197
- CString, 348

appending and concatenating strings, 348

AppendToParagraph

- NTextEdit, 544

application cleanup, 9

array, two-dimensional, 331

attributes of a text editing system, 196, 198, 202, 489, 532, 533, 543

attributes, window, 175

AutoScroll

- CScroller, 289

autoscrolling support, 531

B

background color, setting, 80

Bottom

 CRect, 261

brush color, setting, 81

brush pattern, setting, 81

brush properties, setting, 81

BuildView

 CView, 436

BuildWindow, 314

 CDocument, 65, 68

 CShellDoc, 313

 CTaskDoc, 376

buttons, 210

C

C++ compiler, 153, 221, 224

CApplication, 7–20, 29, 248, 310, 376

 controlling the application, 10

 handling application shutdown, 10

 handling start-up activities for the application, 10

 setting the application's global environment, 16

CArc, 20–28, 304

 drawing, 22

 finding the ending angle, 23

 finding the starting angle, 23

 finding whether the arc has a fill, 23

 setting angles of an arc, 23

 setting the fill, 23

 sizing an arc, 23

 translating angles to points on screen, 23

casting, 153, 221, 224, 331

CBoss, 7, 29–34, 49, 53, 64, 77, 394

CButtonIcon, 35–42, 131, 297

CCentimeterUnits, 401

CCharacterUnits, 399

 setting the base font, 400

CCircle, 43–48, 229

CControlWireFrame, 211

CCTaskWin

 disabling the task window from receiving events, 381

CDesktop, 49–52

 adding a window, 51

 finding whether a window is in the desktop, 51

 getting the front window, 50

 getting the number of windows, 50

 placing a window, 50

 setting the front window, 50

CDialog, 53–63, 174

 activating a dialog, 58

 closing a dialog window, 55

 deactivating a dialog, 59

 disabling a dialog window, 55

 dispatching control events, 59

 enabling a dialog window to receive events, 55

 finding an XVT Portability Toolkit control on a dialog window, 55

 finding whether a dialog window is enabled, 55

 getting a dialog window's XVT Portability Toolkit window handle, 55

 getting a list of subobjects, 58

 getting the border measurement of a dialog window, 55

 getting the inside measurement of a dialog window, 55

 getting the title of a dialog window, 55

 handling a button event, 56

 handling a check box event, 56

 handling a horizontal scroll event, 57

 handling a keyboard event, 58

 handling a list box event, 56

 handling a list button event, 57

 handling a list edit event, 57

 handling a vertical scroll event, 57

 handling an edit control event, 56

 initializing data structures, 58

 setting the title of a dialog window, 54

- sizing a dialog window, 55
- sizing a dialog window by a given width and height, 59
- CDocument, 29, 50, 51, 64–74, 248, 313, 376
 - adding a window, 73
 - changing a document's font, 67
 - closing a document, 70
 - creating a new document, 70
 - creating a window, 68
 - finding a window, 68
 - finding the number of windows, 68
 - finding whether data needs to be saved, 66
 - getting a document's environment, 67
 - getting a document's ID, 66
 - removing a window, 73
 - setting a document's environment, 67
 - setting a document's ID, 72
 - setting whether data needs to be saved, 66
- CEnvironment, 75–84, 393
 - finding whether DISPLAY is color or monochrome, 82
 - getting an object's background color, 79
 - getting an object's font, 80
 - getting an object's foreground color, 79
 - getting the brush color, 81
 - getting the brush pattern, 81
 - getting the object's drawing mode, 81
 - getting the pen pattern, 80
 - getting the pen width, 81
 - getting the pen's ink color, 81
 - inverting background and foreground colors, 82
- CEnvironment settings
 - an object's background color, 80
 - an object's font, 80
 - an objects foreground color, 79
 - brush color, 81
 - brush pattern, 81
 - brush properties, 81
 - DISPLAY to color or monochrome, 82
 - drawing mode, 81
 - drawing tools, 82
 - pen properties, 80
 - the pen color, 80
 - the pen pattern, 80
 - the pen width, 80
- CError, 85–86
- CFixedGrid, 87–96, 110, 409
 - finding a cell's size, 89
 - finding a column, 89
 - finding a column's width, 90
 - finding a row, 89
 - finding a row's height, 90
 - sizing each cell, 89
 - sizing the grid, 90
- CGlobalClassLib, 248
 - getting a unique ID for an object, 102
 - getting the application's task window, 101
- CGlue, 106–108
 - getting the glue type, 108
 - setting the glue type, 107
- CGrid, 87, 109–125, 357, 409
 - drawing a grid, 112
 - drawing a grid and its contents, 112
 - finding placement policy of a grid, 113
 - finding the deepest subview as an event target, 118
 - finding whether grid lines are visible, 113
 - finding whether objects are clipped to cells, 112
 - finding which view is at a given location, 120
 - fitting a view into a grid, 114
 - getting a column, 115
 - getting a row, 115
 - getting the contents of a cell, 114
 - getting the height of a row, 116
 - getting the number of columns, 114
 - getting the number of rows, 114
 - getting the position of an item, 114
 - getting the size of a cell, 115
 - getting the sizing policy, 116
 - getting the width of a column, 116
 - handling a mouse double event, 118
 - handling a mouse down event, 118

- handling a mouse move event, 118
 - handling a mouse up event, 118
 - hiding a grid, 113
 - inserting an item into a grid, 113
 - maximizing the cells of a grid, 116
 - minimizing the cells of a grid, 116
 - placing views correctly, 119
 - removing a specified view from a grid, 113
 - removing a view from a grid, 119
 - removing an item, 113
 - removing and replacing a view in a grid, 114
 - removing and replacing an item, 114
 - resetting a cell, 120
 - setting the drag point for an item, 120
 - setting the placement policy, 112
 - setting the sizing policy, 117
 - setting whether objects are clipped to cells, 112
 - showing a grid, 113
 - sizing a grid, 115
 - sizing a grid and its views, 116
 - sizing each cell, 116
 - specifying number of rows and columns, 116
 - validating rows and columns, 115
- ChangeFont
 - CApplication, 12
 - CBoss, 31
 - CDocument, 67
 - CTaskWin, 380
 - CWindow, 457
- char*
 - CString, 347
- character strings, 345
- check box, definition of, 481
- check boxes, 210, 252, 297, 514
- CHWireFrame, 126–130
 - moving the mouse, 126
- CIcon, 131–138, 297, 357
 - disabling an icon, 133
 - drawing an icon, 133
 - enabling an icon, 133
 - refreshing the image of an icon on the screen, 134
 - setting the font of the title, 133
 - setting the title, 133
 - sizing icons (disabled), 134
 - updating the size of an icon when the font of its title changes, 134
- CInchUnits, 400
- CIterator, 139–142, 153, 154, 221
 - return the first item, 140
 - return the last item, 140
 - return the next item, 140
 - return the preceding item, 140
 - specify the number of steps backward, 141
 - specify the number of steps forward, 140
- CleanMemory
 - CMem, 172
- Clear
 - CList, 155
 - CNativeList, 182
 - CNativeTextEdit, 198
 - CString, 353
- ClearQueue
 - CPrintMgr, 250
- CLine, 143–150, 304
 - drawing, 144
 - finding whether the line has a beginning arrow, 145
 - finding whether the line has an ending arrow, 145
 - setting beginning and ending arrows, 145
 - sizing, 144
 - sizing a line according to points, 145
- CLink, 150–152
 - getting a pointer to the link's item, 151
 - getting the next link in the list, 150
 - getting the previous link, 151
 - setting the link's item, 151
 - setting the next link in the list, 150
 - setting the previous link, 151
- clipping properties, 357
- CList, 9, 150, 153–157, 346
 - actually inserting an item, 156

- concatenating lists without changing them, 155
- concatenating two lists, 155
- decrementing the internal count, 156
- emptying a list, 155
- ensuring a list is not multiply referenced, 156
- finding how many times an item appears, 155
- finding whether a list is empty, 155
- getting the front link, 156
- incrementing the internal count, 156
- inserting an item, 155
- removing an item, 155
- setting the front link, 156
- CListBox, 158–170, 286
 - deselecting a line of text, 161
 - drawing the list box and its views, 162
 - finding whether a list box can resize itself, 161
 - getting the line number of selected text, 160
 - getting the number of lines, 160
 - inserting a line of text, 160
 - removing a line of text, 160
 - scrolling the text lines, 165
 - selecting a line of text, 160
 - setting the font, 162
 - setting the font of a list box, 164
 - setting the glue type of a list box, 164
 - setting the sizing of a list box and its subviews, 164
 - setting whether a list box can resize itself, 161
 - setting whether a list box is draggable, 164
 - sizing, 162
 - sizing the list box and its views, 162
 - sizing the text lines, 165
 - trapping mouse events, 163
- Close
 - CDialog, 55
 - CNativeView, 216
 - CTaskWin, 381
 - CWindow, 453
- CloseAll
 - CApplication, 15
 - CDocument, 68
 - closing a document's windows, 68
 - CMem, 171–173
 - allocating memory for an object, 172
 - deallocating all allocated memory, 172
 - deallocating memory, 172
 - printing a list of all objects for which memory is allocated, 172
 - CModalDialog, 60–63, 174
 - CModalWindow, 174–180
 - CModelessDialog, 60–63
 - CNativeList, 181, 188
 - clearing out all items, 182
 - getting a list of all the items, 182
 - getting the item at a given position, 182
 - getting the number of items, 182
 - inserting a list of strings, 181
 - inserting a single item, 182
 - removing an item from a given position, 182
 - CNativeSelectList, 181, 188–194, 501
 - deselecting all the items, 189
 - Deselecting an item, 189
 - finding whether a given item is selected, 189
 - getting a list of selected items, 188
 - getting the first selected item, 189
 - getting the number of selected items, 188
 - getting the number of the first selected item, 189
 - selecting all the items, 189
 - selecting an item, 189
 - CNativeTextEdit, 195–209, 488, 531, 542
 - activating a text box, 201
 - appending a line of text to text in the buffer, 204
 - appending a paragraph to text in the buffer, 204
 - appending text to existing text, 197
 - clearing the contents of a text box, 198
 - disabling a text box from receiving events, 201
 - drawing a text box, 201

- enabling a text box to receive events, 206
- finding whether a text box contains text, 198
- getting all selected text, 198
- getting all text in the text box, 197, 203
- getting part of a line, 204
- getting part of a paragraph, 204
- getting the attributes of a text box, 198
- getting the maximum number of characters in a text box, 198
- getting the number of actual characters, 197, 203
- getting the number of characters in part of a paragraph, 205
- getting the number of characters in the selected text, 198
- handling a mouse double event, 201
- handling a mouse down event, 202
- hiding a text box, 201
- limiting the number of characters a text box can contain, 198
- redrawing a text box and scrolling its contents to the top, 199
- resuming the redrawing of a text box after text insertions are made, 199
- selecting all text, 197
- sending a keyboard event to a text edit box, 161, 200
- setting an attribute of a text box, 198
- setting text and replacing any existing text, 197
- setting the font, 200
- setting the glue type of a text box, 201
- shifting the text box's origin by a delta point, 200
- showing a text box, 200
- sizing a text box, 200
- suspending the drawing of a text box while text insertions are made, 199
- validating a keyboard event, 199
- CNativeView, 181, 210–220, 252, 476, 481, 501, 514, 521, 552
 - disabling, 213
 - enabling, 213
 - getting the XVT Portability Toolkit window type, 216
 - hiding, 213
 - receiving events, 211
 - setting dragging to TRUE or FALSE, 214
 - setting sizing to TRUE or FALSE, 213
 - setting the ID, 213
 - setting the origin, 213
 - setting the title, 213
 - setting to a different enclosure, 213
 - showing, 213
 - sizing, 212
- colors, inverting background and foreground, 82
- colors, setting, 79
- comparison operators, strings, 349
- concatenating and appending strings, 348
- conserving memory, 331
- ConsumeDelimiter
 - CString, 353
- control handlers, nested, 56
- CONTROL_INFO, 211, 477, 483, 496, 502, 515, 524
- controlling the application, 7
- controls, 210
 - native list, 181
- converting coordinates from global to local, 265
- converting coordinates from local to global, 265
- converting global to local coordinates, 238
- converting local to global coordinates, 238
- coordinate system conversion methods, 238, 265
- coordinates, 235
- coordinates, converting, 238
- coordinates, converting global to local, 238
- coordinates, converting local to global, 238
- CopyView
 - CView, 436
- COrderedIterator, 221–223, 224, 225, 334
 - getting the first item, 222
 - getting the last item, 222
 - getting the next item, 222
 - getting the previous item, 222

- COrderedList, 154, 221, 224–228
 - finding an item at a position, 226
 - inserting an item, 225
 - removing a specified item, 226
 - removing and replacing an item, 226
 - removing from a position, 226
 - reordering all items, 226
- COval, 229–234, 304
 - drawing an oval, 231
- CPoint, 235–240
 - adding a CPoint to the current one, 237
 - changing the horizontal position, 236
 - changing the vertical position, 236
 - converting a CPoint to an XVT Portability Toolkit PNT, 237
 - converting an XVT Portability Toolkit PNT to a CPoint, 236
 - converting from global to local coordinates, 238
 - converting from local to global coordinates, 238
 - finding whether a CPoint is equal to the current one, 237
 - finding whether a CPoint is not equal to the current one, 237
 - getting the vertical position, 236
 - making two CPoints equal, 237
 - setting, 236
 - subtracting a CPoint from the current one, 237
 - temporarily adding the right point to the left point, 237
 - temporarily converting from global to local coordinates, 238
 - temporarily converting from local to global coordinates, 238
 - temporarily subtracting the right point from the left point, 237
 - temporarily translating a CPoint's coordinates, 238
 - translating a CPoint's coordinates, 238
- CPoint, 236
- CPolygon, 240–247, 276, 304
 - drawing, 242
 - setting the fill, 242
 - shifting the origin of a polygon, 242
 - sizing, 242
 - translating CPoints to coordinates for XVT Portability Toolkit, 243
 - translating CPoints to XVT Portability Toolkit coordinates, 243
- CPrintMgr, 248–251
 - getting a list of views in the print queue, 250
 - getting the print window, 250
 - inserting a view into the printing queue, 249
 - removing a view from the printing queue, 249
 - removing everything from the print queue, 250
 - setting up printing, 250
 - taking care of all printing of data, 250
- CRadioGroup, 252–258, 514
 - adding one button at a time, 253
 - adding several buttons at a time, 254
 - drawing, 255
 - getting the selected button, 255
 - removing a button from a group, 255
 - setting the selected button, 255
- CreateControl
 - CNativeView, 216
- CreatePoints
 - CPolygon, 243
- CreateScrollBars
 - NScrollText, 535
- creating a new document, 13
- creating a new menu, 12
- CRect, 175, 259–268
 - adding a CPoint to a CRect, 265
 - adding a CPoint to a CRect to get a translation, 265
 - adding a CRect to a CPoint, 265
 - converting a CRect into an XVT Portability Toolkit RCT, 263

- converting an XVT Portability Toolkit RCT into a CRect, 263
- converting to global coordinates, 266
- converting to local coordinates, 266
- finding whether a CRect is equal to the current one, 263
- finding whether a CRect is not equal to the current one, 263
- finding whether a point is in the CRect, 262
- finding whether the CRect is empty, 262
- getting a pixel measure of the left side, 261
- getting a temporary value for a CRect as inflated, 262
- getting the height in pixels, 262
- getting the length of the bottom side, 261
- getting the length of the top side, 261
- getting the width in pixels, 262
- inflating, 260
- making two CRects the same size, 263
- resetting the dimensions, 260
- resetting the dimensions taking CPoint coordinates, 260
- resetting the length of the bottom side, 261
- resetting the length of the left side, 261
- resetting the length of the top side, 261
- resizing through a union, 263
- setting the height in pixels, 262
- setting the length of the right side, 261
- setting the width in pixels, 262
- subtracting a CPoint from a CRect, 264, 265
- subtracting to get a temporary intersection, 264
- subtracting to get an intersection, 264
- temporarily converting to global coordinates, 266
- temporarily converting to local coordinates, 266
- temporarily resizing through a union, 264
- temporarily translating a CRect's coordinates from one view to another, 266
- translating a CRect's coordinates from one view to another, 266
- CRect
 - CRect, 263
- CRectangle, 269–275, 304, 338
 - drawing, 271
 - giving the rectangle rounded corners, 271
- CRegularPoly, 240, 276–283
 - getting the number of sides, 278
 - rotating a polygon by a number of degrees, 278
 - setting the number of sides, 278
 - sizing, 278
- CResourceManager, 284
- CScroller, 158, 286–296, 441, 552
 - adding a subview to the scroller's contents, 290
 - autoscrolling, 289
 - finding whether thumbtracking is supported, 289
 - getting a scroller's global origin, 290
 - getting a scroller's list of subviews, 291
 - getting the horizontal line increment in pixels, 288
 - getting the horizontal page increment in pixels, 288
 - getting the vertical line increment in pixels, 288
 - getting the vertical page increment in pixels, 288
 - responding to a horizontal scroll event, 289, 428, 525, 534
 - responding to a vertical scroll event, 161, 289, 428, 525, 534
 - scrolling the contents, 289
 - setting dragging for the scroller and its child views, 290
 - setting sizing for the scroller and its child views, 290
 - setting the horizontal increments, 288
 - setting the range of horizontal scrolling, 291
 - setting the range of vertical scrolling, 291
 - setting the vertical increments, 288
 - setting thumbtracking, 289

- shrinking the virtual region to fit the contents, 289
- sizing a scroller, 291
- CSelectIcon, 131, 297–303
 - deselecting an icon, 299
 - finding whether an icon is selected, 299
 - selecting an icon, 299
- CShape, 20, 143, 229, 240, 269, 304–309, 357
 - drawing, 305
- CShellApp, 7, 310–312
 - creating a shell document, 311
 - instantiating and opening a document, 311
 - setting up menus, 311
- CShellDoc, 64, 313–315
 - creating and initializing a shell window, 313
 - invoking a dialog to open a file, 314
 - propagating commands, 313
- CShellWin, 316–321
 - propagating commands, 317
- CSketchPad, 322–330, 357
 - disabling a sketchpad from receiving events, 326
 - drawing, 326
 - enabling a sketchpad to receive events, 326
 - getting the ending point, 324
 - getting the size of the dragged region, 324
 - getting the sketch type, 324
 - getting the starting point, 324
 - handling a mouse down event, 325
 - handling a mouse up event, 325
 - setting the sketch type, 324
- CSparseArray, 331–337
 - getting the item at a given position, 333
 - getting the number of columns, 333
 - getting the number of rows, 333
 - inserting an item, 332
 - removing an item, 332
 - removing the item at a given position, 332
- CSparseArrayIterator, 331, 332, 333
 - getting the next item, 334
- CSparseColIterator, 332, 335
 - getting the next item, 335
- CSparseRowIterator, 332, 334
 - getting the next item, 335
- CSquare, 269, 338–343
 - sizing, 339
- CSring
 - getting a character at a position, 352
- CStartup, 344
- CStartup.cxx file, 344
- CStartup.cxx file, 310
- CString, 345–356
 - appending a char pointer to a string, 348
 - appending a string to a char pointer, 348
 - appending part of a string, 348
 - cleaning up, 354
 - clearing a string's contents, 353
 - comparing a char pointer with a string, 350
 - comparing a string with a char pointer, 349
 - comparing two strings for lexical equality, 349
 - comparing two strings for lexical inequality, 349
 - converting a string to a char pointer, 347
 - defining the plus-equal operator for concatenating char pointers, 348
 - defining the plus-equal operator for concatenating strings, 348
 - doing a greater-than comparison, 350
 - doing a less-than comparison, 350
 - finding whether the input delimiter is included in the string, 353
 - getting the buffer size in number of characters, 354
 - getting the input delimiter, 353
 - getting the length of actual characters, 353
 - giving a string a buffer size, 354
 - setting and getting positions, 352
 - setting the input delimiter, 353
 - setting whether the input delimiter is included in the string, 353
 - sizing the internal structure, 354
 - temporarily appending the right string to the left, 348

- using the input operator, 353
 - using the output operator, 353
- CString interface, 345
- CSubview, 20, 110, 117, 131, 143, 229, 240, 269, 276, 304, 322, 338, 357–372, 420, 423, 433, 441, 450
 - activating a subview and its child views, 360
 - adding a delta point to a subview's origin, 362
 - adding a subview, 366
 - deactivating a subview and its child views, 360
 - disabling a subview and its child views, 360
 - drawing the subview and its child views, 362
 - enabling a subview and its child views, 360
 - finding a subview by its ID, 365
 - finding the deepest subview as the target of an event, 368
 - finding the deepest view containing a point, 365
 - finding the top view containing a point, 365
 - finding the view to which a keyboard event is directed, 366
 - getting every subview containing a point, 366
 - getting the list of subviews, 366
 - getting the selected view, 365
 - getting the subview's number of child views, 366
 - handling a keyboard event, 367
 - handling a mouse double event, 361
 - handling a mouse down event, 361
 - handling a mouse move event, 361
 - handling a mouse up event, 361
 - hiding a subview and its child views, 359
 - placing the bottom view, 365
 - placing the top view, 365
 - removing a subview, 367
 - setting a view to receive a keyboard event, 366
 - setting the dragging of a subview and its child views, 364
 - setting the environment of a subview and its
 - child views, 363
 - setting the font of a subview and its child views, 364
 - setting the glue of a subview and its child views, 364
 - setting the selected view, 364
 - setting the sizing of a subview and its child views, 364
 - showing a subview and its child views, 359, 367
 - sizing a subview and its child views, 367
- CSwitchBoard, 51, 373–375
 - channelling events to objects, 373
- CSwitchBoard as liason between the XVT Portability Toolkit and XVT-Power++, 373
- CTaskDoc, 376–378, 379
 - building the task window, 376
- CTaskWin, 379–385
 - changing the task window's font, 380
 - closing the task window, 381
 - enabling the task window to receive events, 381
 - hiding, 381
 - showing, 381
 - sizing the task window, 380
- CText, 386–392
 - deselecting and thus uninverting text, 388
 - drawing, 388
 - finding whether text is selected, 388
 - finding whether the text background is opaque, 387
 - getting the height of a text box, 388
 - getting the text, 387
 - getting the width of a text box, 388
 - selecting and thus inverting text, 388
 - setting a title, 387
 - setting the font, 388
 - setting the sizing, 389
 - setting the text, 387
 - setting whether the text background is opaque, 388
- CUnits, 393–403

- getting the base font, 400
- getting the global CUnits object, 399
- getting the logical unit for a physical
 - horizontal mapping, 398
- getting the logical unit for a physical vertical
 - mapping, 398
- getting the output device, 396
- getting the owner of a CUnits object, 396
- mapping logical horizontal units to physical
 - units, 397
- mapping logical vertical units to physical
 - units, 397
- setting the global units, 398
- setting the horizontal and vertical printer
 - mappings, 397
- setting the horizontal and vertical screen
 - mappings, 397
- setting the horizontal printer mapping, 397
- setting the horizontal screen mapping, 397
- setting the output device, 396
- setting the owner of units for screen/printer
 - mappings, 395
- setting the vertical printer mapping, 397
- setting the vertical screen mapping, 397
- updating the owner of a CUnits object, 399
- cut, paste, copy operations, 195
- CVariableGrid, 87, 110, 409–419
 - getting a column, 411
 - getting a row, 411
 - getting the height of a row, 412
 - getting the size of a cell, 411
 - getting the width of a column, 411
 - maximizing or minimizing a column, 413
 - maximizing or minimizing a row, 413
 - resetting the default height for columns, 412
 - resetting the default height of the grid
 - internally, 413
 - resetting the default width for rows, 412
 - resetting the default width of the grid
 - internally, 413
 - resetting the size of a grid internally, 413
 - sizing a column, 412
 - sizing a grid, 412
 - sizing a row, 412
 - specifying the number of rows and columns, 413
- CView, 29, 54, 195, 197, 210, 248, 269, 322, 357, 359, 360, 376, 379, 386, 420–440, 450, 465, 471, 476, 488, 542
 - activating a view, 423
 - activating a view and its subviews, 423
 - adding a delta point to a view's origin, 429
 - adding a delta point to a view's origin and
 - those of its subviews, 429
 - building a view, 436
 - copying a view, 436
 - deactivating a view, 423
 - deactivating a view and its subviews, 423
 - disabling a view, 423
 - disabling a view and its subviews, 424
 - drawing, 426
 - drawing a view and its subviews, 426
 - drawing for the printer, 434
 - enabling a view, 423
 - enabling a view and its subviews, 424
 - finding a subview containing a point, 435
 - finding the deepest subview as the target of
 - an event, 435
 - finding the deepest subview containing a
 - point, 434
 - finding whether a view is active, 423
 - finding whether a view is draggable, 431
 - finding whether a view is enabled, 423
 - finding whether a view is sharing an
 - environment, 430
 - finding whether a view is sizable, 432
 - finding whether a view is visible, 422
 - getting a region in coordinates local to a
 - given view, 429
 - getting a view's command, 432
 - getting a view's double command, 433
 - getting a view's enclosure, 433
 - getting a view's environment, 429
 - getting a view's glue type, 431

- getting a view's origin, 429
- getting a view's origin relative to the window, 429
- getting a view's title, 433
- getting a view's window, 432
- getting the coordinates of the visible part of a clipped view, 428
- getting the ID of a view, 432
- getting the region of a view in global coordinates, 428
- getting the region of a view in relative coordinates, 428
- getting the selected view, 435
- getting the wire frame, 432
- handling a keyboard event, 427
- handling a mouse double event, 425
- handling a mouse double event and passing it on, 425
- handling a mouse down event, 424
- handling a mouse down event and passing it on, 425
- handling a mouse move event, 424
- handling a mouse move event and passing it on, 425
- handling a mouse up event, 425
- handling a mouse up event and passing it on, 425
- hiding a view, 422
- hiding a view and its subviews, 423
- making a view and its subviews visible, 422
- notifying a view to print itself, 433
- preparing a view for drawing, 436
- propagating commands, 434
- sending a keyboard event to a window, 427
- setting a view to a different enclosure, 433
- setting a view's command, 432
- setting a view's double command, 433
- setting a view's environment, 67, 430
- setting a view's environment and that of its subviews, 430
- setting a view's glue type, 431
- setting a view's title, 433
- setting the dragging of a view, 431
- setting the dragging of a view and its subviews, 432
- setting the font, 427
- setting the font of a view and its subviews, 428
- setting the glue type of a view and its subviews, 431
- setting the ID of a view, 432
- setting the sizing of a view, 432
- setting the sizing of a view and its subviews, 432
- setting the wire frame, 432
- showing a view, 422
- sizing, 427
- sizing a view and its subviews, 427
- CVirtualFrame**, 286, 357, 441–449
 - adding a subview, 444
 - drawing the visible region, 443
 - enlarging the virtual area to include a region, 443
 - getting the dimensions of the virtual region, 443
 - scrolling the contents, 444
 - setting the height and width of the virtual region, 443
 - setting the horizontal scrolling range, 445
 - setting the scrolling origin, 442
 - setting the vertical scrolling range, 445
 - shrinking the virtual region to fit its contents, 443
 - sizing the virtual region, 443
- CVWireFrame**, 404–408
 - moving the mouse, 404
- CWindow**, 65, 174, 316, 357, 379, 450–464
 - activating a window internally, 458
 - changing the font, 457
 - closing a window, 453
 - deactivating a window internally, 458
 - disabling a window from receiving events, 455
 - drawing a window's background, 455

- enabling a window to receive events, 455
 - finding whether a window has a background, 454
 - finding whether a window is closable, 454
 - finding whether a window is sizable, 454
 - finding whether a window is using a private environment, 456
 - getting a window's document, 454
 - getting an XVT Portability Toolkit window handle, 454
 - getting the coordinates of the visible part of a clipped view, 456
 - handling a menu bar command, 456
 - handling a native view event, 457
 - hiding, 455
 - internally handling a horizontal scrolling event, 459
 - internally handling a vertical scrolling event, 459
 - propagating commands, 453
 - sending a keyboard event to a window, 455
 - setting a window's background, 454
 - setting a window's enclosure(disabled), 455
 - setting the selected view, 456
 - setting the title, 456
 - showing, 455
 - sizing, 455
 - sizing internally, 457
 - updating the menu bar, 453
 - CWireFrame, 126, 404, 431, 465–475
 - adding a delta point to a wire frame's origin, 469
 - autoscrolling, 471
 - deselecting(undrawing), 468
 - drawing, 469, 470
 - drawing a wire frame at a different coordinate, 470
 - drawing the wire frame's resizing handles, 470
 - finding whether a wire frame can receive dragging events, 468
 - finding whether a wire frame can receive sizing events, 467
 - finding whether a wire frame is currently selected, 468
 - finding whether the dragging area contains a given point, 471
 - finding whether the sizing area contains a given point, 471
 - getting the owner, 468
 - handling a dragging event, 467
 - handling a mouse down event, 468
 - handling a mouse move event, 469
 - handling a mouse up event, 469
 - handling a sizing event, 467
 - resizing to encompass a point, 470
 - selecting(drawing), 468
 - sizing, 469
- D**
- data, getting access to, 64
 - DeActivate
 - CSubview, 360
 - Deactivate
 - CView, 423
 - NListEdit, 509
 - debugging utility, 171
 - Decrement
 - CList, 156
 - DeDeactivate
 - CView, 423
 - DeleteParagraph
 - NTextEdit, 544
 - DeSelect
 - NCheckBox, 483
 - Deselect
 - CSelectIcon, 299
 - CText, 388
 - CWireFrame, 468
 - DeselectAll
 - CNativeSelectList, 189
 - DeselectItem
 - CNativeSelectList, 189
 - DeselectLine
 - CListBox, 161

- desktop, 7
- DestructiveHandle
 - CList, 156
 - CString, 354
- dialog behavior, 53
- dialog control handlers
 - handling a button event, 56
 - handling a check box event, 56
 - handling a horizontal scroll event, 57
 - handling a keyboard event, 58
 - handling a list button event, 57
 - handling a list edit event, 57
 - handling a radio button event, 56
 - handling a vertical scroll event, 57
 - handling an edit control event, 56
- dialog event handlers
 - handling a list box event, 56
- dialogs
 - controls, 54
 - modal, 53
 - modeless, 53
 - XVT Portability Toolkit functionality, 56
- Disable
 - CDialog, 55
 - CIcon, 133
 - CNativeTextEdit, 201
 - CNativeView, 213
 - CSketchPad, 326
 - CTaskWin, 381
 - CView, 423
 - CWindow, 455
 - NWinScrollbar, 554
- disabling the other windows on the screen, 51, 174
- DoAboutBox
 - CApplication, 12, 13
- DoActivate
 - CDialog, 58
 - CView, 423
 - CWindow, 458
- DoAutoScroll
 - CWireFrame, 471
- DoBorderCheck
 - CWireFrame, 471
- DoButton
 - CDialog, 56
- DoCheckBox
 - CDialog, 56
- DoClose
 - CApplication, 11, 13
 - CDocument, 69, 70
- DoCommand, 297, 481
 - CApplication, 11
 - CBoss, 30
 - CDocument, 65, 68
 - CScroller, 290
 - CShellApp, 311
 - CShellDoc, 313
 - CShellWin, 317
 - CSketchPad, 322
 - CView, 434
 - CWindow, 453
- DoControl
 - CDialog, 59
 - CWindow, 457
- DoCreate
 - CDialog, 58
- DoDeactivate
 - CDialog, 59
 - CSubview, 360
- DoDeactivateWindow
 - CWindow, 458
- DoDisable
 - CSubview, 360
 - CView, 424
- DoDraggingCheck
 - CWireFrame, 471
- DoDraw
 - CGrid, 112
 - CListBox, 162
 - CSubview, 362
 - CView, 426
- DoEdit
 - CDialog, 56
- DoEnable

- CSubview, 360
- CView, 424
- DoHide
 - CSubview, 359
 - CView, 423
- DoHit
 - CNativeview, 211
 - NButton, 477
 - NCheckBox, 483
 - NListBox, 496
 - NListButton, 502
 - NListEdit, 509
 - NRadioButton, 515
 - NScrollBar, 524
- DoHScroll
 - CDialog, 57
 - CWindow, 459
- DoKey
 - CApplication, 14
 - CDialog, 58
 - CDocument, 71
 - CSubview, 367
 - CView, 427
- DoListBox
 - CDialog, 56
- DoListButton
 - CDialog, 57
- DoListEdit
 - CDialog, 57
- DoMenuCommand
 - CApplication, 11
 - CBoss, 30
 - CDocument, 69
 - CWindow, 456
- DoModal, 51
- DoMouseDouble
 - CSubview, 361
 - CView, 425
- DoMouseDown
 - CSubview, 361
 - CView, 425
- DoMouseMove
 - CSubview, 361
 - CView, 425
- DoMouseUp
 - CSubview, 361
 - CView, 425
- DoNew
 - CApplication, 12, 13
 - CDocument, 68, 70
 - CShellApp, 311
 - CShellDoc, 314
- DoOpen, 68
 - CApplication, 11, 13
 - CShellApp, 311
 - CShellDoc, 314
- DoPageSetup
 - CApplication, 12, 14
 - CDocument, 69, 70
- DoPlaceView
 - CGrid, 120
- DoPrint, 248
 - CApplication, 12, 14
 - CDocument, 69, 70
 - CPrintMgr, 250
 - CView, 433
- DoPrintDraw
 - CView, 433
- DoQuit, 11
- DoRadioButton
 - CDialog, 56
- DOS, 171
- DoSave
 - CApplication, 11
 - CDocument, 69, 70
- DoSaveAs, 14
 - CApplication, 11, 14
 - CDocument, 69, 70
- DoSet
 - CSubview, 364
- DoSetDragging
 - CListBox, 164
 - CScroller, 290
 - CSubview, 364

- CView, 432
- DoSetEnvironment
 - CSubview, 363
 - CView, 430
- DoSetFont
 - CListBox, 164
 - CView, 428
- DoSetGlue
 - CSubview, 364
 - CView, 431
- DoSetOrigin
 - CSubview, 362
 - CView, 429
- DoSetSizing
 - CListBox, 164
 - CScroller, 290
 - CSubview, 364
 - CView, 432
- DoShow
 - CSubview, 359
 - CView, 422
- DoSize
 - CGrid, 116
 - CListBox, 162
 - CSubview, 367
 - CView, 427
- DoVScroll
 - CDialog, 57
 - CWindow, 459
- Drag
 - CWireFrame, 470
- dragging and stretching views, 431
- Draw
 - CArc, 22
 - CGrid, 112
 - CIcon, 133
 - Cline, 144
 - CNativeTextEdit, 201
 - COval, 231
 - CPolygon, 242
 - CRadioGroup, 255
 - CRectangle, 271

- CShape, 305
- CSketchPad, 326
- CText, 388
- CView, 426
- CVirtualFrame, 443
- CWindow, 455
- CWireFrame, 469
- DrawFrameGrabbers
 - CWireFrame, 470
- drawing mode, setting, 81
- drawing tools, setting up, 82
- DrawWireFrame
 - CWireFrame, 470
- DumpMemory
 - CMem, 172

E

- EM_ALL, XVT Portability Toolkit, 58
- EM_NONE, XVT Portability Toolkit, 58
- Enable
 - CDialog, 55
 - CIcon, 133
 - CNativeTextEdit, 206
 - CNativeView, 213
 - CSketchPad, 326
 - CTaskWin, 381
 - CView, 423
 - CWindow, 455
 - NWinScrollBar, 554
- EnlargeToFit
 - CVirtualFrame, 443
- environment
 - native list, 181
- environment settings
 - CArc, 20, 322
 - CCircle, 43
 - CFixedGrid, 87
 - CIcon, 131
 - CLine, 143
 - CListBox, 158
 - CNativeSelectList, 188
 - CNativeTextEdit, 195
 - CNativeView, 210

- COval, 229
- CPolygon, 240
- CRectangle, 269
- CRegularPoly, 276
- CScroller, 286
- CSelectIcon, 297
- CShellWin, 316
- CSquare, 338
- CText, 386
- CVariableGrid, 409
- NLineText, 488
- NListBox, 495
- NListButton, 501
- NListEdit, 507
- NScrollText, 531
- NTextEdit, 542
- NWinScrollBar, 552
- environment settings, XVT Portability Toolkit, 75
- error reporting utility, 85
- event filtering, 358
- event mask constants, XVT Portability Toolkit, 58
- F**
- File menu, 11
- File menu item events, 13
- Find
 - COrderedList, 226
- FindDeepSubview
 - CGrid, 118
 - CSubview, 365
 - CView, 434
- FindDocument, 15
- FindEventTarget
 - CListBox, 163
 - CSubview, 368
 - CView, 435
- finding a window, 68
- FindSubview
 - CSubview, 365
 - CView, 435
- FindSubviews
 - CSubview, 366
- FindWindow
 - CDesktop, 51
 - CDocument, 68
- FindXVTControl
 - CDialog, 55
- First
 - CIterator, 140
 - COrderedIterator, 222
- font, setting, 80
- G**
- G pointer, 30
- G->SetPrintMgr(), 251
- Get
 - CString, 352
- GetAllItems
 - CNativeList, 182
- GetApplication
 - CGlobalClassLib, 101
- GetAttributes
 - CNativeTextEdit, 198
- GetBackgroundColor
 - CEnvironment, 79
- GetBaseFont
 - CUnits, 400
- GetBrushColor
 - CEnvironment, 81
- GetBrushPattern
 - CEnvironment, 81
- GetBufferSize
 - CString, 354
- GetCellSize
 - CFixedGrid, 89
 - CGrid, 115
 - CVariableGrid, 411
- GetCharSize
 - CUnits, 400
- GetClippedFrame
 - CView, 428
 - CWindow, 456
- GetCol
 - CFixedGrid, 89
 - CGrid, 115
 - CVariableGrid, 411

- GetColNumber
 - CVariableGrid, 414
- GetCommand
 - CView, 432
- GetContents
 - CGrid, 114
 - CSparseArray, 333
 - GetContents, 114
- GetCWindow
 - CView, 432
- GetDeselectCommand
 - NCheckBox, 483
- GetDesktop
 - CGlobalClassLib, 100
- GetDocument
 - CWindow, 454
- GetDoubleCommand
 - CView, 433
- GetDrawingMode
 - CEnvironment, 81
- GetDynamicMapping
 - CUnits, 396
- GetEnclosure
 - CView, 433
- GetEndingAngle
 - CArc, 23
- GetEndPoint
 - CSketchPad, 324
- GetEnvironment, 15
 - CDocument, 67
 - CView, 429
 - CWindow, 456
- GetFirstSelectedItem
 - CNativeSelectList, 189
- GetFont
 - CEnvironment, 80
- GetForegroundColor
 - CEnvironment, 79
- GetFrame
 - CDialog, 55
 - CView, 428
- GetFrontLink
 - CList, 156
- GetFrontWindow
 - CDesktop, 50
- GetFullPar
 - CNativeTextEdit, 204
- GetGlobal
 - CPoint, 238
 - CRect, 266
- GetGlobalFrame
 - CDialog, 55
 - CView, 428
- GetGlobalOrigin
 - CScroller, 290
 - CView, 429
- GetGlobalUnits
 - CUnits, 399
- GetGlue, 108
 - CView, 431
- GetHeight
 - CFixedGrid, 90
 - CGrid, 116
 - CText, 388
 - CVariableGrid, 412
- GetHitItem
 - CGrid, 120
- GetHLineIncrement
 - CScroller, 288
 - NScrollText, 534
- GetHPageIncrement
 - CScroller, 288
 - NScrollText, 534
- GetHPrinterMapping
 - CUnits, 397
- GetHScreenMapping
 - CUnits, 397
- GetId
 - CDocument, 66
 - CGlobalClassLib, 102
 - CView, 432
- GetInflatedRect
 - CRect, 262
- GetInputDelimiter

- CString, 353
- GetItem
 - CLink, 151
 - CNativeList, 182
- GetKeyFocus
 - CSubview, 366
- GetKeyFocusCommand
 - NListEdit, 509
- GetKeyFocusLostCommand
 - NListEdit, 509
- GetLimit
 - CNativeTextEdit, 198
- GetLine
 - NTextEdit, 545
- GetLineIncrement
 - NScrollBar, 524
- GetLineInternal
 - CNativeTextEdit, 204
- GetLocal
 - CPoint, 238
 - CRect, 266
- GetLocalframe
 - CView, 429
- GetMargin
 - CNativeTextEdit, 199
- GetMaxPosition
 - NScrollBar, 523
- GetMinPosition
 - NScrollBar, 523
- GetNCharInLine
 - NTextEdit, 545
- GetNCharInPar
 - NTextEdit, 545
- GetNCharInPartPar
 - CNativeTextEdit, 205
- GetNCharInSelection
 - CNativeTextEdit, 198
- GetNCharInternal
 - CNativeTextEdit, 203
- GetNCharInText
 - CNativeTextEdit, 197
- GetNext
 - CLink, 150
- GetNLineInPar
 - NTextEdit, 545
- GetNLineInSelection
 - NTextEdit, 545
- GetNLineInText
 - NTextEdit, 545
- GetNParInSelection
 - NTextEdit, 545
- GetNParInText
 - NTextEdit, 545
- GetNumberOfLines
 - CListBox, 160
- GetNumberOfSides
 - CRegularPoly, 278
- GetNumCols
 - CGrid, 114
 - CSparseArray, 333
- GetNumDocuments, 15
- GetNumItems
 - CNativeList, 182
- GetNumRows
 - CGrid, 114
 - CSparseArray, 333
- GetNumSelectedItems
 - CNativeSelectList, 188
- GetNumViews
 - CSubview, 366
- GetNumWindows
 - CDesktop, 50
 - CDocument, 68
- GetOrigin
 - CView, 429
- GetOutputDevice
 - CUnits, 396
- GetOwner
 - CUnits, 396
 - CWireFrame, 468
- GetPageIncrement
 - NScrollBar, 524
- GetParagraph
 - NTextEdit, 544

- GetPartLine
 - CNativeTextEdit, 204
- GetPartPar
 - CNativeTextEdit, 204
- GetPenColor
 - CEnvironment, 81
- GetPenPattern
 - CEnvironment, 80
- GetPenWidth
 - CEnvironment, 80
- GetPlacement
 - CGrid, 113
- GetPosition
 - CGrid, 114
- GetPostion
 - NScrollBar, 523
- GetPrevious
 - CLink, 151
- GetPrintWindow
 - CPrintMgr, 250
- GetQueue
 - CPrintMgr, 250
- GetRow
 - CFixedGrid, 89
 - CGrid, 115
 - CVariableGrid, 411
- GetRowNumber
 - CVariableGrid, 413
- GetSBType
 - NWinScrollBar, 554
- GetScrollingOrigin
 - CVirtualFrame, 442
- GetSelectCommand
 - NCheckBox, 483
- GetSelectedItems
 - CNativeSelectList, 188
- GetSelectedLine
 - CListBox, 160
- GetSelectedText
 - CNativeTextEdit, 197
- GetSelectedView
 - CSubview, 365
- CView, 435
- GetSelectPosition
 - CNativeSelectList, 189
- GetSizingPolicy
 - CGrid, 116
- GetSketchedRegion
 - CSketchPad, 324
- GetSketchType
 - CSketchPad, 324
- GetSomeText
 - NTextEdit, 546
- GetStartingAngle
 - CArc, 23
- GetStartPoint
 - CSketchPad, 324
- GetSubObjects
 - CDialog, 58
 - CSubview, 15, 71, 366
 - CView, 434
- GetSubviews
 - CScroller, 291
 - CSubview, 366
- GetTaskWin
 - CGlobalClassLib, 101
- GetText
 - , 197
 - CText, 387
- GetTextCommand
 - NListEdit, 509
- GetTextInternal
 - CNativeTextEdit, 203
- GetTitle
 - CDialog, 55
 - CView, 433
- GetTranslated
 - CPoint, 238
 - CRect, 266
- GetVirtualFrame
 - CVirtualFrame, 443
- GetVLineIncrement
 - CScroller, 288
 - NScrollText, 534

- GetVPageIncrement
 - CScroller, 288
 - NScrollText, 534
- GetVPrinterMapping
 - CUnits, 397
- GetVScreenMapping
 - CUnits, 397
- GetWidth
 - CFixedGrid, 90
 - CGrid, 116
 - CText, 388
 - CVariableGrid, 411
- GetWireFrame, 431
 - CView, 432
- GetXVTType
 - CNativeView, 216
- GetXVTWindow
 - CDialog, 55
 - CWindow, 454
- giving a window modal behavior, 174
- global class library, 33
- global CUnits object, 393
- global environment, 7
- global environment object, 75, 77
- Globalize
 - CPoint, 238
 - CRect, 266
- globally accessible objects/variables, 17
- globals
 - user-specified, 33
- Glue
 - CView, 431
- glue types, 106
- gluing subviews, 364
- Go, 344
 - CApplication, 10
- grid cell operations, 114
- grid cell, definition of, 109
- grid characteristics, 109
- grid coordinates, 110
- grid placement methods, 113, 119
- grid sizing methods, 116
- grid snapping, 117
- grids
 - clipping, 111
 - dragging state, 111
 - mouse events, 111
 - placement, 111
 - snapping point, 111
 - visibility of, 111
 - with cells of homogeneous size, 87
 - with cells of varying size, 409
- grids, adjusting, 110
- grids, placement, 110
- grids, sizing policies, 110, 111
- Grow
 - CString, 354
- GU pointer, 9, 30
- H**
- H
 - CPoint, 236
- handling events affecting the entire application, 7
- HasBeginningArrow
 - CLine, 145
- HasEndingArrow
 - CLine, 145
- Height
 - CRect, 262
- Hide
 - CNativeTextEdit, 201
 - CNativeView, 213
 - CTaskWin, 381
 - CView, 422
 - CWindow, 455
 - NWinScrollBar, 553
- HideGrid
 - CGrid, 113
- HLogicalToPhysical
 - CUnits, 397
- HPhysicalToLogical
 - CUnits, 398
- HScroll, 521
 - CScroller, 289, 428, 525, 534

I

- icon resource IDs, 131
- icon resource, a definition, 131
- icon resources, 284
- icons
 - adding button behavior, 35
- inating, 101
- Increment
 - CList, 156
- indexing strings, 352
- Inflate
 - CRect, 260
- InList
 - CList, 155
- Insert
 - CGrid, 113, 114
 - CList, 155
 - CNativeList, 181, 182
 - COrderedList, 225
 - CPrintMgr, 249
 - CSparseArray, 332
- InsertInternal
 - CList, 156
- InsertLine
 - CListBox, 160
- interface between all objects and events, 373
- InvertColors
 - CEnvironment, 82
- IsActive
 - CView, 423
- IsBackgroundDrawn
 - CWindow, 454
- IsClipping
 - CGrid, 112
- IsClosable
 - CWindow, 454
- IsColorOn
 - CEnvironment, 82
- IsDraggable
 - CView, 431
 - CWireFrame, 468
- IsEmpty
 - CList, 155
 - CNativeTextEdit, 198
- IsEnabled
 - CDialog, 55
 - CView, 423
- IsEnvironment
 - CView, 430
- IsFilled
 - CArc, 23
 - CPolygon, 242
- IsGridVisible
 - CGrid, 113
- IsHeightSelfAdjusted
 - CListBox, 161
- IsItSelected
 - CNativeSelectList, 189
- IsNull
 - CRect, 262
- IsOpaque
 - CText, 387
- IsPointInRect
 - CRect, 262
- IsSelected
 - CSelectIcon, 299
 - CText, 388
 - CWireFrame, 468
 - NCheckBox, 482
- IsSizable
 - CView, 432
 - CWindow, 454
 - CWireFrame, 467
- IsTerminating, 101
- IsThumbtracking
 - CScroller, 289
- istream&
 - CString, 353
- IsUpdatingTextEdits, 101
- IsVisible
 - CView, 422
- iterating order, 333
- iterating over lists, 139, 221

K

keeping track of windows, 49

Key

CNativeTextEdit, 200

CView, 427

CWindow, 455

keyboard events, 71

L**Last**

CIterator, 140

COrderedIterator, 222

lectAll, 189

Left

CRect, 261

Length

CString, 353

link pointers, 150

list button, a definition, 57

list edit, a definition, 57

lists, iterating order, 153

lists, order, 221

lists, ordered, 224

lists, removing multiply-stored items, 153

Localize

CPoint, 238

CRect, 266

LogicalToPhysical

CUnits, 398

M

Macintosh, 51, 53, 174, 379

macro, PWRAssert, 85

macros, CUnits, 394

main(), 10, 310, 344

making variables/objects globally accessible, 17

malloc, XVT Portability Toolkit function, 171

managing an application's workspace/window layout, 49

managing windows, 64

mapping values, 394

mapping XVT Portability Toolkit events to XVT-Power++ methods, 373

masking dialog events, 58

MEM_DEBUG option, 171

memory allocation, 171

memory management, 171

memory, conserving, 331

menu bar, 11, 12

menubar, 69

modal dialogs, a definition, 53

modal windows, 174

modeless dialogs, a definition, 53

Motif, 284, 379

mouse event methods, scroller, 299

mouse events, button icons, 37

mouse events, text editing, 201

mouse events, views, 424

mouse methods, CListBox, 163

mouse methods, subviews, 360

mouse methods, wire frames, 468

mouse, operating, 210

MouseDown

CButtonIcon, 37

CGrid, 118

CListBox, 164

CNativeTextEdit, 201

CSelectIcon, 299

CView, 425

MouseDown

CButtonIcon, 37

CGrid, 118

CListBox, 163

CNativeTextEdit, 202

CSelectIcon, 299

CSketchPad, 325

CView, 424

CWireFrame, 468

MouseMove

CButtonIcon, 37

CGrid, 118

CHWireFrame, 126

CListBox, 163

CView, 424

CVWireFrame, 404

- CWireFrame, 469
- MouseUp, 163
 - CButtonIcon, 37
 - CGrid, 118
 - CSketchPad, 322, 325
 - CView, 425
 - CWireFrame, 469
- MS-Windows, 171
- multi-line text, 542
- multiple insertion of objects, 224
- multiple references, lists, 153
- N**
- native list controls, 181
- native look and feel, 476, 481, 514, 521
- native views, 210
- native window manager, 210, 476, 481, 514, 521
- NativeHeight
 - NScrollBar, 524
- NativeWidth
 - NScrollBar, 524
- NButton, 210, 476–480
 - receiving events, 477
- NCheckBox, 210, 481–487
 - deselecting, 483
 - finding whether a check box is selected, 482
 - getting the number of the deselect command, 483
 - getting the number of the select command, 483
 - receiving events, 483
 - selecting, 483
 - setting the number of the deselect command, 483
 - setting the number of the select command, 483
 - setting the sizing, 483
 - setting the title, 483
- NeedsSaving
 - CDocument, 66
- nested control handlers, 56
- nested views, 357

Next

- CIterator, 140
- COrderedIterator, 222
- CSparseArrayIterator, 334
- CSparseColIterator, 335
- CSparseRowIterator, 335
- NLineText, 195, 386, 488–494
 - internally resetting the height after a font change, 490
 - Setting attributes of text boxes, 489
 - setting the font, 490
 - sizing, 490
- NListBox, 188, 495–500
 - receiving events, 496
- NListButton, 188, 501–506
 - receiving events, 502
- NListEdit, 181, 507–513
 - activating, 509
 - deactivating, 509
 - getting the key focus command, 509
 - getting the key focus lost command, 509
 - getting the text command, 509
 - receiving events, 509
 - setting keyboard focus commands, 508
 - setting text commands, 509
- NRadioButton, 210, 514–520
 - receiving events, 515
 - setting the title, 515
- NScrollBar, 210, 441, 521–530, 552
 - getting the maximum thumb position, 523
 - getting the minimum thumb position, 523
 - getting the optimum native height of a scroll bar, 524
 - getting the optimum native width of a scroll bar, 524
 - receiving events, 524
 - setting the line increment, 524
 - setting the line increment and page increment, 523
 - setting the range of thumb movement, 523
 - setting the thumb position, 523
- NScrollBar

- setting the page increment, 524
 - NScrollText, 531–541, 542, 552
 - getting the horizontal line increment, 534
 - getting the horizontal page increment, 534
 - getting the vertical line increment, 534
 - getting the vertical page increment, 534
 - internally creating the scroll bar(s), 535
 - setting the font, 535
 - setting the horizontal increments for line and page increments, 533
 - setting the vertical increments for line and page increments, 534
 - sizing, 535
 - updating the scroll bars after a scroll, 535
 - NTextEdit, 195, 386, 531, 542–552
 - adding a paragraph, 544
 - appending a paragraph, 544
 - deleting a paragraph, 544
 - getting a line, 545
 - getting a paragraph, 544
 - getting some text, 546
 - getting the number of characters in a line, 545
 - getting the number of characters in a paragraph, 545
 - getting the number of characters in selected text, 545
 - getting the number of characters in the text, 545
 - getting the number of lines in a paragraph, 545
 - getting the total number of lines, 545
 - selecting a line, 545
 - selecting a paragraph, 544
 - selecting some text, 546
 - setting a line, 545
 - setting a paragraph, 544
 - NULLIcon, 284
 - NumItems
 - CList, 155
 - NWinScrollBar, 552–557
 - disabling from receiving events, 554
 - enabling to receive events, 554
 - getting the scroll bar's type, 554
 - hiding, 553
 - setting the enclosure, 554
 - setting the ID, 553
 - setting the title, 553
 - showing, 553
 - sizing a window's scroll bar, 553
- O**
- one-line text entries, 488
 - opening a document, 13
 - operator-
 - CPoint, 237
 - CRect, 264, 265
 - operator delete
 - CMem, 172
 - operator new
 - CMem, 172
 - operator!=
 - CPoint, 237
 - CRect, 263
 - CString, 349, 350
 - operator+
 - CList, 155
 - CPoint, 237
 - CRect, 264, 265
 - CString, 348
 - operator+=
 - CList, 155
 - CPoint, 237
 - CRect, 263, 265
 - CString, 348
 - operator<
 - CString, 350, 351
 - operator<=
 - CString, 352
 - operator-=
 - CPoint, 237
 - CRect, 264
 - operator=
 - CPoint, 237
 - CRect, 263
 - operator==

- CPoint, 237
- CRect, 263
- CString, 349
- operator>
 - CString, 350
- operator>=
 - CString, 351
- operator[]
 - CString, 352
- order in lists, 153
- order, iterating, 333
- ordered lists, 224
- ostream&
 - CString, 353
- output device, 394
- P**
 - paragraph methods, 544
 - XVT-Power++ desktop, 450
 - PWRAssert macro, 85
 - PWRNoError option, 85
 - pen color, setting, 80
 - pen pattern, setting, 80
 - pen properties, setting, 80
 - pen width, setting, 80
 - PhysicalToLogical
 - CUnits, 398
 - pixel mapping, 393
 - pixels, 32, 393
 - PlaceBottomSubview
 - CSubview, 365
 - placement methods for grids, 119
 - placement, grids, 110
 - PlaceTopSubview
 - CSubview, 365
 - PlaceView
 - CGrid, 119
 - PlaceWindow
 - CDesktop, 50
 - platforms, icon resources for, 284
 - PNT
 - CPoint, 237
 - pointers, link, 150
 - Prepare
 - CView, 436
 - Presentation Manager, 379
 - Previous
 - CIterator, 140
 - COrderedIterator, 222
 - print job title, 249
 - PRINT_RCD, 249
 - PrintDraw
 - CView, 434
 - printer mappings, 393
 - printer units, 32, 393
 - printing from an XVT-Power++ application, 248
 - printing queue, 248
 - printing queue methods, 249
 - printing.XVT Portability Toolkit, 250
 - PrintThread
 - CPrintMgr, 250
 - providing access to global objects and data, 29
 - Q**
 - queue, printing, 248
 - R**
 - radio buttons, 210, 252, 514
 - RCT
 - CRect, 263
 - reference counting, 171
 - reference counting, CString, 345
 - Refresh
 - CIcon, 134
 - Remove
 - CGrid, 113
 - CList, 155
 - CNativeList, 182
 - COrderedList, 226
 - CPrintMgr, 249
 - CSparseArray, 332
 - RemoveButton
 - CRadioButton, 255
 - RemoveDocument, 17
 - RemoveLine
 - CListBox, 160

- RemoveSubview
 - CGrid, 119
 - CSubview, 367
- RemoveWindow
 - CDesktop, 51
 - CDocument, 73
- Renumber
 - COrderedList, 226
- Replace
 - CGrid, 114
 - COrderedList, 226
- Reset
 - CNativeTextEdit, 199
- ResetCell
 - CGrid, 120
- ResetFrame
 - CVariableGrid, 413
- ResetHeight
 - CVariableGrid, 413
 - NLineText, 490
- ResetWidth
 - CVariableGrid, 413
- ReSize
 - CWireFrame, 470
- resource IDs, icons, 131
- resource manager, 284
- Resume
 - CNativeTextEdit, 199
- Right
 - CRect, 261
- Rotate
 - CRegularPoly, 278
- rubberband frame, 465
- S**
 - saving data under a different name, 14
 - screen mappings, 393
 - scroll bars, 210
 - scroll bars, thumbtracking, 286
 - ScrollLines
 - CListBox, 165
 - ScrollTextCallback
 - NScrollText, 535
 - ScrollViews
 - CScroller, 289
 - CVirtualFrame, 444
 - searching for a document, 15
 - SectRect
 - CRect, 260
 - Select
 - CSelectIcon, 299
 - CText, 388
 - CWireFrame, 468
 - NCheckBox, 483
 - SelectAll
 - CNativeSelectList, 189
 - selection box, 297
 - SelectItem
 - CNativeSelectList, 189
 - SelectLine
 - CListBox, 160
 - NTextEdit, 545
 - SelectParagraph
 - NTextEdit, 544
 - SelectSomeText
 - NTextEdit, 546
 - SelectText
 - CNativeTextEdit, 197
 - SelfAdjustHeight
 - CListBox, 161
 - Set
 - CString, 352
 - SetAngles
 - CArc, 23
 - SetArrows
 - CLine, 145
 - SetAttribute
 - CNativeTextEdit, 198
 - NLineText, 489
 - SetBackgroundColor
 - CEnvironment, 80
 - SetBackGroundDrawing
 - CWindow, 454
 - SetBaseFont
 - CCharacterUnits, 400

- SetBrush
 - CEnvironment, 81
- SetBrushColor
 - CEnvironment, 81
- SetBrushPattern
 - CEnvironment, 81
- SetBufferSize
 - CString, 354
- SetClipping
 - CGrid, 112
- SetColor
 - CEnvironment, 79
- SetColorOn
 - CEnvironment, 82
- SetCommand
 - CView, 432
- SetDefaultHeight
 - CVariableGrid, 412
- SetDefaultWidth
 - CVariableGrid, 412
- SetDesktop
 - CGlobalClassLib, 100
- SetDeslectCommand
 - NCheckBox, 483
- SetDevelopmentMetrics
 - CUnits, 396
- SetDoubleCommand
 - CView, 433
- SetDragging
 - CNativeView, 214
 - CView, 431
 - CWireFrame, 467
- SetDragPoint
 - CGrid, 120
- SetDrawingEnv
 - CEnvironment, 82
- SetDrawingMode
 - CEnvironment, 81
- SetDrawingPoints
 - CArc, 23
- SetDynamicMapping
 - CUnits, 396
- SetEnclosure
 - CNativeView, 213
 - CView, 433
 - CWindow, 455
 - NWinScrollBar, 554
- SetEnvironment
 - CDocument, 67
 - CNativeTextEdit, 201
 - CView, 67, 430
- SetFilled
 - CArc, 23
 - CPolygon, 242
- SetFocusCommands
 - NListEdit, 508
- SetFont
 - CEnvironment, 80
 - CIcon, 133
 - CListBox, 162, 164
 - CNativeTextEdit, 200
 - CText, 388
 - CView, 427
 - NLineText, 490
 - NScrollText, 535
- SetForegroundColor
 - CEnvironment, 79
- SetFrontLink
 - CList, 156
- SetFrontWindow
 - CDesktop, 50
- SetGlobalUnits
 - CUnits, 398
- SetGlue, 107
 - CListBox, 164
 - CNativeTextEdit, 201
 - CView, 431
- SetHIncrements
 - CScroller, 288
 - NScrollText, 533
- SetHScrollRange
 - CScroller, 291
 - CVirtualFrame, 445
- SetId

- CDocument, 72
- CNativeView, 213
- CView, 432
- NWinScrollBar, 553
- SetIncrements
 - NScrollBar, 523
- SetInputDelimiter
 - CString, 353
- SetItem
 - CLink, 151
- SetKeyFocus
 - CSubview, 366
- SetLimit
 - CNativeTextEdit, 198
- SetLine
 - NTextEdit, 545
- SetMargin
 - CNativeTextEdit, 198
- SetNext
 - CLink, 150
- SetNumberOfSides
 - CRegularPoly, 278
- SetNumCells
 - CGrid, 116
 - CVariableGrid, 413
- SetOpaque
 - CText, 388
- SetOrigin
 - CNativeTextEdit, 200
 - CNativeView, 213
 - CPolygon, 242
 - CView, 429
 - CWireFrame, 469
- SetOutputDevice
 - CUnits, 396
- SetOwner
 - CUnits, 395
- SetParagraph
 - NTextEdit, 544
- SetPen
 - CEnvironment, 80
- SetPenColor
 - CEnvironment, 80
- SetPenPattern
 - CEnvironment, 80
- SetPenWidth
 - CEnvironment, 80
- SetPlacement
 - CGrid, 112
- SetPoint
 - CPoint, 236
- SetPosition
 - NScrollBar, 523
- SetPrevious
 - CLink, 151
- SetPrinterMapping
 - CUnits, 397
- SetRange
 - NScrollBar, 523
- SetRect
 - CRect, 260
- SetRoundedCorners
 - CRectangle, 271
- SetSave
 - CDocument, 66
- SetScreenMapping
 - CUnits, 397
- SetScrollingOrigin
 - CVirtualFrame, 442
- SetSelectCommand
 - NCheckBox, 483
- SetSelectedButton
 - CRadioGroup, 255
- SetSelectedView
 - CSubview, 364
 - CWindow, 456
- SetSizing
 - CIcon, 134
 - CNativeView, 213
 - CText, 389
 - CView, 432
 - CWireFrame, 467
 - NCheckBox, 483
- SetSizingPolicy

- CGrid, 117
- SetSketchType
 - CSketchPad, 324
- SetText
 - CNativeTextEdit, 197
 - CText, 387
- SetTextCommand
 - NListEdit, 509
- SetThumbtracking
 - CScroller, 289
- setting a document's ID, 71
- setting an object's colors, 79
- setting DISPLAY to color, 82
- setting the font for the global application
 - environment, 12
- setting the font of a list box and its subviews, 164
- setting up menus, 12
- SetTitle
 - CDialog, 54
 - CIcon, 133
 - CNativeView, 213
 - CText, 387
 - CView, 433
 - CWindow, 456
 - NCheckBox, 483
 - NRadioButton, 515
 - NWinScrollBar, 553
- SetUnits, 393
- SetUpMenus, 12
 - CShellApp, 311
- SetVIncrements
 - CScroller, 288
 - NScrollText, 534
- SetVirtualFrame
 - CVirtualFrame, 441, 443
- SetVScrollRange
 - CScroller, 291
 - CVirtualFrame, 445
- SetWireFrame, 431
 - CView, 432
- Show
 - CNativeTextEdit, 200
 - CNativeView, 213
 - CTaskWin, 381
 - CView, 422
 - CWindow, 455
 - NWinScrollBar, 553
- ShowGrid
 - CGrid, 113
- ShrinkToFit
 - CVirtualFrame, 443
- ShrinkToFit
 - CScroller, 289
- Shutdown
 - CApplication, 10
- Size
 - Arc, 23
 - CCircle, 44
 - CDialog, 55
 - CGrid, 115
 - CIcon, 134
 - CLine, 144, 145
 - CListBox, 162
 - CNativeTextEdit, 200
 - CNativeView, 212
 - CPolygon, 242
 - CRegularPoly, 278
 - CScroller, 291
 - CSquare, 339
 - CVariableGrid, 412
 - CView, 427
 - CVirtualFrame, 443
 - CWindow, 455
 - CWireFrame, 469
 - NLineText, 490
 - NScrollText, 535
 - NWinScrollBar, 553
- SizeCell
 - CFixedGrid, 89
 - CGrid, 116
 - CVariableGrid, 412
- SizeCol
 - CVariableGrid, 412
- SizeLines

- CListBox, 165
- SizeOwner, 108
- SizeRow
 - CVariableGrid, 412
- SizeWindow
 - CDialog, 59
 - CTaskWin, 380
 - CWindow, 457
- sizing policies for grids, 110
- sizing policies, grids, 111
- sketching area, 322
- specifying screen locations, 235
- spreadsheet, 409
- Startup
 - CApplication, 10
- static methods for global unit object settings, 398
- static text drawing, 386
- StepBack
 - CIterator, 141
- StepForward
 - CIterator, 140
- stickiness properties, 106
- storage capacity of a string, 345
- storing a set of screen coordinates, 259
- stretching a view, 106
- string comparison operators, 349
- string conversion operator, 345
- string input/output methods, 353
- string storage, 345
- string storage and safety, 345
- strings, indexing, 352
- Suspend
 - CNativeTextEdit, 199
- T**
- task window, 376, 379
- templates, 153, 221, 224, 331
- terface, 432
- text editing mouse events, 201
- text editing system, attributes of, 196, 198, 202, 489, 532, 533, 543
- text editing, autoscrolling support, 531
- text entries, one-line, 488
- text properties, 195
- text, color settings, 386
- text, organizing into paragraphs and lines, 542
- thumbtracking, scroll bars, 286
- title of print job, 249
- Top
 - CRect, 261
- transferring control to XVT-Power++ from main, 344
- Translate
 - CPoint, 238
 - CRect, 266
- type of glue, 106
- U**
- UNIX, 284
- UpdateMenus
 - CApplication, 12
 - CDocument, 71
 - CWindow, 453
- UpdateOwner
 - CUnits, 399
- UpdateSize
 - CIcon, 134
- UpdateTextEdits, 101
- URL, 10, 57, 58
- URL resource file, 54
- URL resource format, 53
- URL resources, 174
- user-specified globals, 33
- utility classes
 - CEnvironment, 75
 - CError, 85
- V**
- V
 - CPoint, 236
- Validate
 - CGrid, 115
 - CNativeTextEdit, 199
- VLogicalToPhysical
 - CUnits, 397
- void pointers, 153, 221, 224, 331

VPhysicalToLogical

CUnits, 398

VScroll, 521

CScroller, 161, 289, 428, 525, 534

Wwide interface, 359, 360, 421, 422, 423, 427, 428,
429, 430, 431, 432, 434

Width

CRect, 262

window attributes, 175

window manager, 450

window manager, native, 210, 481, 514, 521

window types, XVT Portability Toolkit, 450

Windows, 379

windows

keeping track of, 49

managing, 64

setting the active window, 51

setting the front window, 50

updating, 64

windows, modal, 174

X

XVT Portability Toolkit, 10, 181, 188, 243

XVT Portability Toolkit and printing, 250

XVT Portability Toolkit attributes

WSF_HSCROLL, 552

WSF_VSCROLL, 552

XVT Portability Toolkit button event, 56

XVT Portability Toolkit dialog functionality, 56

XVT Portability Toolkit events, 373

XVT Portability Toolkit interface to edit controls,
56XVT Programmer's Guide, 75, 196, 198, 202,
211, 237, 263, 373, 450, 477, 483, 489, 490,
496, 502, 515, 524, 532, 533, 543

XVT Programmer's Reference, 248

XVT Portability Toolkit window attributes, 175

XVT Portability Toolkit window types, 450